MODELING AND CONTROL OF A SELF-ASSEMBLED ROBOTIC SWIMMER

BY

CARLOS ALBERTO ORDUNO

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Mechanical Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2012

Urbana, Illinois

Adviser:

Professor Timothy Bretl

ABSTRACT

This thesis presents a control strategy based on model learning for a self-assembled robotic "swimmer". The swimmer forms when a liquid suspension of ferro-magnetic micro-particles and a non-magnetic bead are exposed to an alternating magnetic field that is oriented perpendicular to the liquid surface [1]. It can be steered by modulating the frequency of the alternating field. We model the swimmer as a unicycle and learn a mapping from frequency to forward speed and turning rate using locally-weighted projection regression. We apply iterative linear quadratic regulation with a receding horizon to track motion primitives that could be used for path following. Hardware experiments validate our approach.

A mis padres, con amor y admiración

ACKNOWLEDGMENTS

Numerous persons made valuable contributions to this work. I wish to acknowledge their support, without which it would not have been possible:

- My adviser, Professor Timothy Bretl, who provided me the opportunity to work on this fascinating project. Your guidance and motivation was crucial in the completion of it, and your creativity for approaching unsolved problems, an inspiration.
- Aaron Becker who aided enormously throughout the project and my graduate studies, and provided me with inspiring ideas. Your love for robotics is contagious!
- The RMS group, including Miles Johnson and Cem Onyuksel, who consistently provided a platform for testing ideas.
- Hugo León, who helped constructing the experiment setup and provided great support when starting on this project.
- The University of Illinois College of Engineering, the SURGE office and the MechSE department, who provided me with the financial means to complete my degree.
- My family, who consistently encouraged me and whose love and values defined who I am.
- My wife, Carolina, whose endless support, love and patience was the key to the completion of my graduate degree. Your engineering mindset was an example for me. ... this part of my life has been incredibly fulfilling because of you.

To all of you: I sincerely thank you!

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

DAC	Digital to Analog Converter
FM	Frequency Modulator
GPR	Gaussian Process Regression
iLQR	iterative Linear Quadratic Regulator
LWR	Locally Weighted Regression
LWPR	Locally Weighted Projection Regression
LQ	Linear Quadratic
LQR	Linear Quadratic Regulator
LOC	Linear Optimal Control
MPC	Model Predictive Control
OC	Optimal Control
SVR	Support Vector Regression

CHAPTER 1

INTRODUCTION

Recent work has shown that a suspension of ferro-magnetic micro-particles (45μm) will form a large-scale structure when exposed to an alternating magnetic field that is oriented perpendicular to the liquid surface [1–5].

This structure becomes mobile and turns into a self-assembled robotic "swimmer" when symmetry is broken by adding a larger non-magnetic bead (1mm). The motion of this swimmer is unidirectional, generally toward the bead that forms the head. Changes in frequency of the alternating magnetic field (30-40Hz) cause changes in the arrangement and characteristic length of the swimmer, which in turn change the surrounding flow and affect subsequent motion.

Previous work has focused on understanding the physics that govern these phenomena. In this thesis we focus on developing a control strategy that allows us to steer the resulting swimmer along primitives of canonical shape (e.g., circles of different radius) that could be used to follow arbitrary paths.

We model the self-assembled robotic swimmer as a nonholonomic unicycle, under the constraint that it only moves forward. We apply locally weighted projection regression (e.g., see [6]) to learn the mapping from applied frequency to forward speed and turning rate given data collected offline. We apply iterative linear quadratic regulation (e.g., see [7]) with a receding horizon (e.g., see [8]) to track motion primitives given visual feedback. We validate our approach in hardware experiments with the system shown in Figure 1.1.

Our work is motivated by applications that include targeted drug delivery (e.g. see [9]) and non-invasive surgery (e.g. see [10]). The self-assembled robotic system that we describe here (and, in particular, the visual feedback that we currently require) is clearly not appropriate for these applications yet. We view this system as a platform for the development of new control strategies and for the exploration of new mechanisms for self-assembly, which we hope may build a foundation for future systems with more practical application.

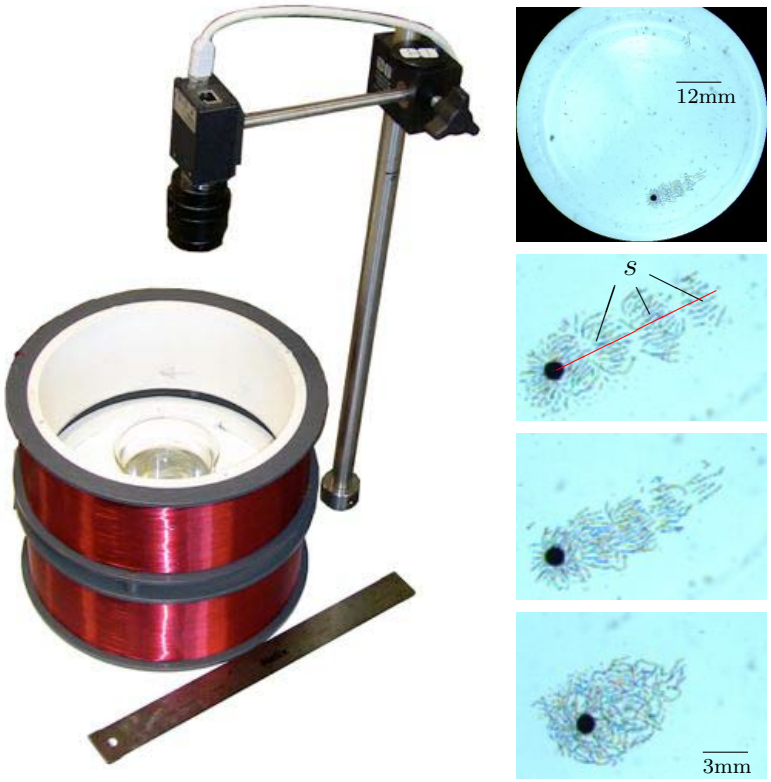

Figure 1.1: Our hardware platform (left) and the resulting self-assembled robotic "swimmer" (right), which arises when an alternating magnetic field is applied perpendicular to an air-water interface that contains a suspension of ferro-magnetic micro-particles and a non-magnetic bead. The structure of the swimmer changes with the frequency of the alternating field—shown is the same swimmer at a frequency of 28, 40, and 77Hz. Also, the swimmer segments are indicated by s and the centerline by a red line.

1.1 Challenges and Contributions

The objective and main contribution of this thesis is to present a hardware and software platform together with the algorithm and mathematical framework that enables control of a self-assembled swimmer [1].

The main challenge originates from the self-assembly process. Each swimmer assembles into a structure with particular physical characteristics, such as segment width, number of segments and head position with respect to the centerline. The movement of the head depends on these characteristics and therefore, changes between swimmers.

A generalized model for the swimmer is hard to obtain analytically, since the system is subject to changes and the dynamics depend on the self-assembly process. Therefore we use

real-time data to learn a local model using Locally Weighted Learning, specifically Locally Weighed Projection (LWPR) Regression.

Over the long term, changes on the system dynamics are expected reorganization of the nickel chains. Therefore we continuously update the LWPR model. The obtained model provides a low error forward simulation where future states are predicted based on past observations. This model provides a good approximation over a horizon of 5s (see chapter 6).

The control task is defined as trajectory tracking and the controller is obtained using an algorithm called iterative Linear Quadratic Regulator (iLQR). iLQR is capable of iteratively improving an initial guessed control input. After a number of iterations, the output of iLQR is a control policy $\boldsymbol{u}_k = \pi(\boldsymbol{x}_k)$, locally optimal with respect to a defined cost function which penalizes the deviation from a desired trajectory, i.e. the set of states that define a circular motion.

In iLQR, the desired trajectory need not be feasible, i.e. there need not be some control input to achieve our desired sequence of states. But instead it improves over an initial guess and minimizes the tracking error between the current location and the desired one. The initial guess is based on the first curvature value, κ (see [11]), of the target circle and the swimmer response to a particular frequency. We also use a receding horizon in iLQR to improve the controller performance.

In summary we break the tasks into the following parts:

- System Identification or Model Learning. Learns a model of the system from system measurements by using Locally Weighted Projection Regression (LWPR).
- Trajectory Optimization and Tracking. Generates a locally optimal trajectory and provides a closed loop control law $\boldsymbol{u}_k = \pi(\boldsymbol{x}_k)$ by designing an iterative Linear Quadratic Regulator (iLQR).
- On-line Adaptation. Accounts for regular plant changes by adapting the LWPR model on-line.

Task		Solution
Stage	Challenge	Method/Algorithm
System Identification	Self-assembly process	LWPR
Trajectory Tracking	Modeling errors and noise	iLQR
Model Adaptation	Changes on swimmer structure	MPC/LWPR

Table 1.1: The different stages of the platform we propose for controlling the swimmer together with the identified challenges and proposed solutions.

1.2 Outline

This thesis proceeds as follows. Chapter 2 gives a brief overview of related work, focusing on methods of self-assembly, of magnetic control of micro-robots, and of model learning. Chapter 3 describes both the self-assembled robotic swimmer and our hardware implementation in more detail. Chapter 4 presents our approach to model learning and validation, which is based on the use of locally weighted projection regression. Chapter 5 presents our approach to control, which is based on the use of iterative linear quadratic regulation with a receding horizon. Chapter 6 shows our experimental results. Chapter 7 synthesizes our approach and concludes with a brief discussion of opportunities for future work.

CHAPTER 2

RELATED WORK

The swimmers covered by this thesis exhibit self-assembly. Their size and actuation mechanisms place them in the class of magnetically controlled micro robots. Finally, their dynamics are dependent on their structure, which is randomly generated when they are formed. This uncertainty requires model learning before we can control them.

2.1 Self-assembly

Magnetic swimmers fit into a broader category of self-assembled systems, an overview of which is given in [12]. In that work by Whitesides and Grzybowski [3], self-assembly was defined as *a reversible process by which pre-existing discrete entities bind to each other without being directed externally.* The magnetic swimmers form spontaneously under appropriate particle loading and magnetic frequency. The process is reversible in that the swimmer components quickly separate out when the driving frequency is removed.

Self-assembly is of considerable interest to the robotics community in community, from applications such as self-arrangement in parts-handling [13] to task-oriented self-assembly of modular robots [14]. Our swimmers have no autonomy, so only their formation can be considered self-assembly. With external control, these swimmers can be directed to follow paths, go from one point to another, impact their environment, or grow into larger swimmers. The former tasks are related to parts-handling while the latter is similar to modular robotics.

2.2 Magnetically controlled micro robots

Magnetically controlled micro robots are an active field of research. Because they involve actuation from a distance they are being pursued for applications in medical devices [15–17].

Sudo et al. presented a 5mm magnet with a flexible tail, propelled by an external magnetic field [15], capable of swimming in viscous fluid. This robot was designed to be navigable through the human heart and large arteries. Abbott et al. contrast the efficacy of pulling

using a magnetic gradient verse micro robots that flex or use helical propeller to swim through a fluid, and find that swimming micro robots become more desirable as distance from the generating magnetic field increases and as the robot size decreases. The swimmers presented by [16, 17] exhibit variation in velocity, but are constrained to all have the same orientation, so they cannot be steered independently.

Parallels can also be found with the non-swimming micro manipulation work of Diller and Sitti et al. They controlled the 2D coordinates of multiple micro-scale permanent magnets by exploiting heterogeneity in the dimensions of the magnets [18,19]. In this work a single control signal was applied to each magnet, but unlike the helical swimmers of [16,17], independent control was possible due to heterogeneity. Similarly, the magnetic swimmers we present have unique dynamic models that are based on their structure. This heterogeneity leads to unique forward velocity, turning rate and first curvature κ (see section 5.3.3) as a function of magnetic frequency, and could enable independent control.

2.3 Model Learning for Control

Robotic systems can contain nonlinearities on its dynamics. In which case modeling assumptions are made to simplify the process, but leading to inaccuracies. Approaching these type of systems using exclusively analytical first principle models, results on a control performance that rapidly decreases due to difficult to these nonlinearities, such as friction and backlash. In this situation, estimating the dynamics model from measured data and improving the prediction performance as new system observations are integrated, poses an interesting alternative.

The essential idea behind model learning is to present the modeling problem as a regression problem in which now the task is to learn the mapping function from inputs to outputs ($\boldsymbol{f} : \boldsymbol{x} \mapsto y$). On system control, several relationships can be defined such as forward model, inverse model, policy and value function depending on how the input/output pairs are defined.

Several regression methods have been explored for the task of creating a dynamical model from a sampled system such as Support Vector Regression (SVR) [20], Gaussian Process Regression (GPR) [21] and local methods such as Locally Weighted Projection Regression (LWPR) [6], among other non-parametric regression techniques. A few of these have been applied to control complex robotic systems such as stick juggling robots [22], legged robots [23], helicopters [8], biped walking robot [24], anthropomorphic arms [25] and humanoid robots [26].

CHAPTER 3

SYSTEM DESCRIPTION

The self-assembled swimmers consist of nickel micro-particles suspended on the liquid-air interface. Driven by a collective behavior and due to an external alternating magnetic field, these particles congregate into a snake-like structure capable of generating water flows on the surface.

In the following section we describe the dynamics behind the self-assembly process, the steady-state behavior and their response to changes on the magnetic field. This work will lead us to the design of a modeling algorithm as described on chapter 4. A brief description of the hardware platform used is also presented.

3.1 Self-Assembled Magnetic Swimmers

Magnetic particles suspended on the liquid-air interface, subject to an alternating magnetic field, will exhibit a self-assembly behavior [4]. The phenomenon is driven by the collective response to the magnetic field and the generation of surface waves by the oscillating particles.

The self-assembly process starts when in the presence of the alternating field (20 to 120Hz) the magnetic moments from the particles are aligned along the chain direction driven by dipole-dipole interactions [4]. On a larger scale, the chains order to form segments, which in contrast to the chains, have an antiferromagnetic ordering.

The applied alternating field also causes the particles to oscillate in place dragging the adjacent water. Consequently the segment oscillations create fluid motion at both ends of the snake nearly equal for frequency values below 85Hz [1]. By placing a glass bead (1-1.5mm in diameter) near one end, it will attach and the liquid displacement will be higher at the opposite tail producing a net forward displacement.

Belkin and Snezhko present in-depth experimental and theoretical studies of the water flows [2] which range from 0.4cm/s to 2cm/s.

Self-assembled swimmers present a system for the manipulation of objects on the surface of water. By adjusting the driving signal frequency we are capable of modifying the segment

arrangement and characteristic length. As the frequency value is increased the segment sections contract increasing the overall water flow. With the presence of a glass bead on one end, an increase on the magnetic frequency results on an increase in the differential flow between both sides resulting on a net movement towards the end of less flow. This inhomogeneity opens the opportunity for doing limited swimmer steering by selectively choosing a frequency value to achieve a particular turning rate or swimming curvature.

The feasible curvatures followed by a swimmer depend on the symmetric property of the structure. Given a perfectly symmetric swimmer, varying the control frequency will only result on a change of the forward speed with no modification of the turning rate.

Below critical frequency values [5] the swimmer is stable in the short-term, subject only to change when chains detach from their corresponding segment due to collision with the beaker wall, and interactions with the water flow and suspended particles along the path. These changes modify the response of the swimmer to a given frequency.

We select the 30 to 40Hz frequency range for driving the swimmers since this offers two relevant advantages: we can assume the swimmer configuration is reversible and this range exhibits the largest changes in velocity and turning rate.

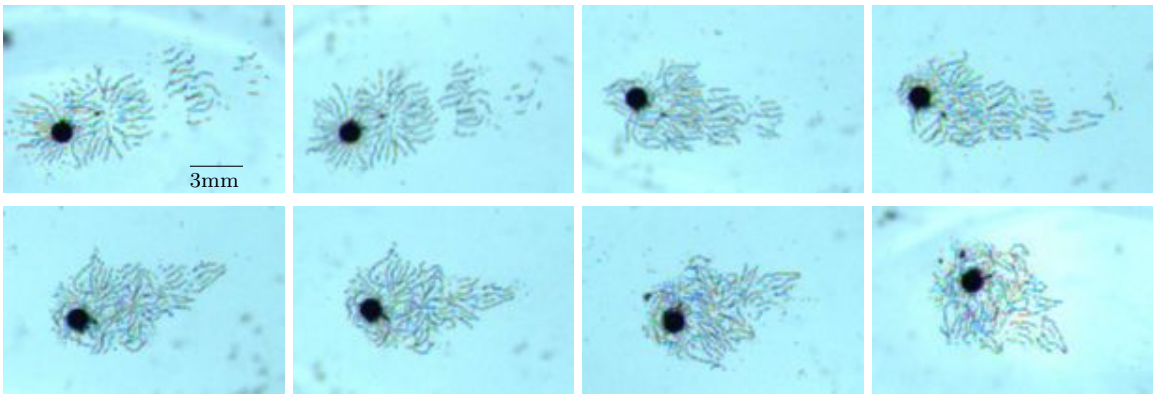

Figure 3.1: A self-assembled swimmer shown at magnetic frequency 24,30,40,50,60,70,80 and 90Hz. The swimmer begins with three well-defined links, but changes shape significantly.

3.2 Hardware Platform

Our hardware platform consists of a cylindrical glass container (145mm in diameter) inside a Helmholtz coil (220mm in diameter, 2x 83mm in height). The coil is energized by an alternating voltage generated by a dedicated frequency modulator (FM) circuit connected

to an amplifier. The control system runs on a workstation (Intel Xeon 2.4 GHz). The desired frequency is commanded to the FM circuit via an external DAC box (UEI Power DNA-A0-308-350). The control loop is closed using a digital camera (camera: Basler A601f / lens: Edmund Industrial Optics 35mm double gauss 54689). A backlight is added using a flat illumination screen obtained from a portable computer.

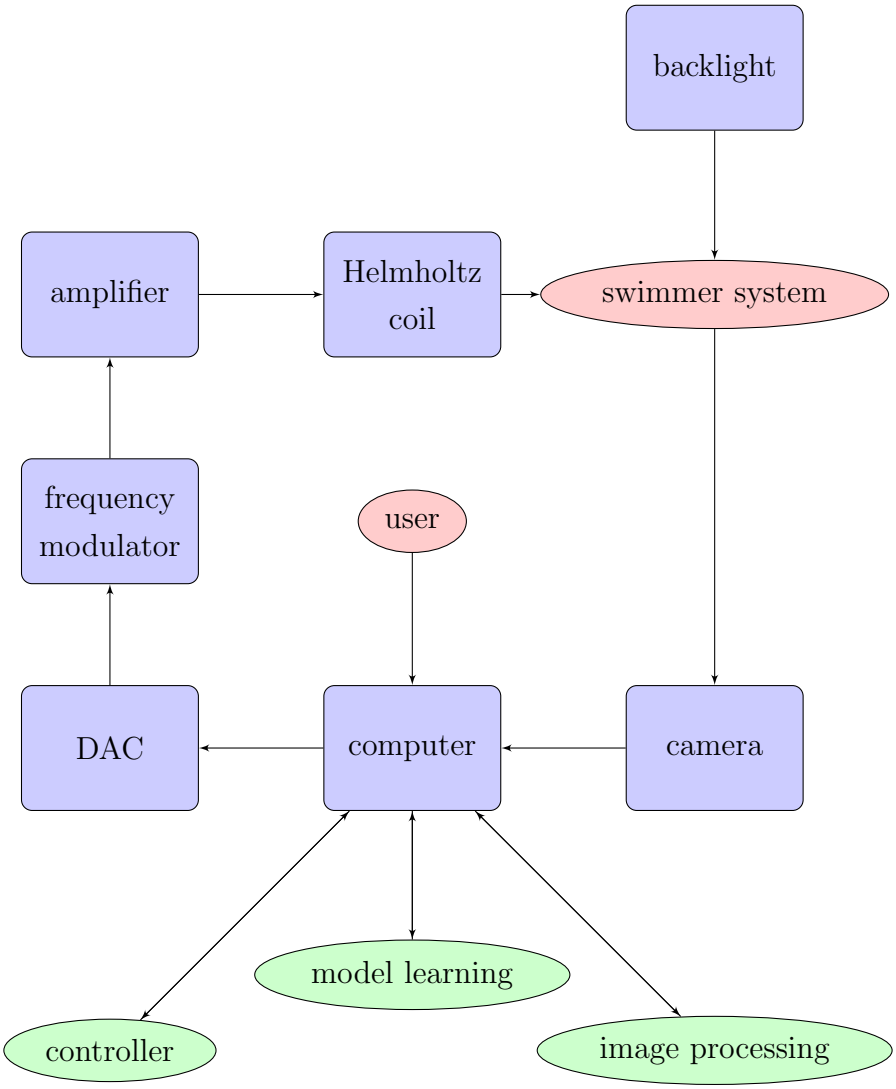

Figure 3.2: Block diagram of experimental setup. Hardware components are shown using blue blocks and software components using green ellipses.

CHAPTER 4

MODEL LEARNING

The swimmer structural configuration obtained by the formation process defines its dynamics. Given this uncertainty, our ability of modeling the swimmer reliably beforehand is limited. We could attempt to obtain a model whose parameters depend on the observed features of the swimmer, such as number of segments, segment width, particle concentration, but this approach would require both complex image processing tasks to obtain these important features and an expensive high-resolution vision system. Instead we choose to use simpler computer vision algorithms to locate the swimmer head, record a history of these positions and use them to learn an online forward dynamic model, where future swimmer states can be predicted based on past observations.

We have presented on chapter 2 various function approximation or regression methods which have been used for the control of robotic systems. Were seek one for the task of learning a forward model and for selecting one, we have set ourselves the following criteria:

1. requires no previous knowledge of the swimmer model dynamics structure
2. can achieve fast computation both during learning and evaluation
3. is capable of on-line learning

These criteria imply we are looking for an incremental regression method that is structurally adaptive and is based on local models. We choose Locally Weighted Projection Regression (LWPR). This is a regression method that builds local linear regressions of a non-linear function and can operate on-line.

The remainder of this chapter outlines the details of the LWPR method and presents the corresponding algorithm.

4.1 Locally Weighted Projection Regression

Locally Weighted Projection Regression (LWPR) (see [6, 27]) is an extension of Receptive Field Weighted Regression in [28]. LWPR approximates a nonlinear function by piecewise

linear models. The learning process consists of obtaining the number linear models N and characteristic parameters $\mathbf{b}_n$ of the hyperplanes that describe each local model. A crucial step is determining the region of validity, also called Receptive Field (RF), in which the local model can be trusted.

The receptive field of each linear model can be computed from a Gaussian kernel:

$$w_n = exp\left(-\frac{1}{2}\left(\boldsymbol{x} - \mathbf{c}_n\right)^\top \mathbf{D}_n \left(\boldsymbol{x} - \mathbf{c}_n\right)\right) \tag{4.1}$$

where w_n is a weight, $\boldsymbol{x}$ is the query point , $\mathbf{c}_n$ is the center of the n-th linear model, and $\mathbf{D}_n \geq 0$ is a distance metric that determines the size and shape of region of validity of the linear model. Other kernel functions are possible.

In LWPR local models are built continuously in the entire support area of the input data at selected points in input space. The prediction for a query point $\hat{y}$ is then formed as the weighted average of the predictions of the local models whose receptive fields are selected. If N local models are chosen, the prediction is calculated by a weighted average and the regression function can be written as:

$$\hat{y} = \hat{f}\left(\boldsymbol{x}\right) = \frac{\sum_{n=1}^{N} w_n \hat{y}_n}{\sum_{n=1}^{N} w_n} \tag{4.2}$$

$$\hat{y}_n = b_n^0 + \boldsymbol{b}_n^\top \left(\boldsymbol{x} - \mathbf{c}_n\right) \tag{4.3}$$

where b_n^0 and $\boldsymbol{n}_k^\top$ denote the offset and slope of the n-th local linear model.

During the learning process, both the shape of the receptive fields $\mathbf{D}_n$ and the parameters $\mathbf{b}_n$ of the local models are adjusted such that the error between the predicted values and the observed targets is minimal. The regression parameter $\mathbf{b}_n$ can be computed incrementally and online using the partial least squares method [29]. The distance matrix $\mathbf{D}_n$ determines the size and shape of each local model and can be learned through gradient descent given by:

$$\mathbf{d}^{n+1} = \mathbf{d}^n - r\frac{\partial V}{\partial \mathbf{d}} \tag{4.4}$$

where r is the learning rate for gradient descent, $\mathbf{D} = \mathbf{d}^\top \mathbf{d}$ and V is a cost function which addresses the issue of over-fitting data [30]. The number of receptive fields is also adapted. Receptive fields are created if for a given training data point, no existing receptive field possesses a weight w_i, that is greater than a threshold value of w_{ten}. Similarly, if two local models produce a weight greater than a threshold w_{prune}, the model whose receptive field is

smaller is pruned.

The computational speed of LWPR is not affected by the number of training points since they are not explicitly stored but rather encoded into local linear models.

LWPR can adapt to changes of the system dynamics in real-time as new data becomes available. This is done by setting a forgetting factor λ, which is selected to balance between preserving the learned model and adapting to new data.

We use the LWPR algorithm implementation provided by Klanke et al. [31].

4.2 Model Linearization

System dynamics linearization is a considerable element of the computational cost during on-line control since calculation of the system dynamic's derivatives $\frac{\partial f}{\partial \mathbf{x}}$ and $\frac{\partial f}{\partial \mathbf{u}}$ most be computed along an entire trajectory multiple times for the iLQR algorithm (see chapter 5). One way to estimate the derivatives is by finite differences i.e. querying multiple points close to where the linearization is about and using the result to estimate the derivative.

Analytical derivatives of the LWPR model are also possible, and on implementation significant improvement in speed of up to $5\times$, was obtained when using analytical derivatives.

Differentiating (4.2) with respect to the input $\boldsymbol{x}$ and substituting (4.1) yields:

$$\begin{aligned}
\frac{\partial \hat{f}(\boldsymbol{x})}{\partial \boldsymbol{x}} &= \frac{1}{\sum_{n=1}^{N} w_n} \sum_{n=1}^{N} \left(\frac{\partial w_n}{\partial \boldsymbol{x}} \hat{y}_n + w_n \frac{\partial \hat{y}_n}{\partial \boldsymbol{x}} \right) \\
&\quad - \frac{1}{\left(\sum_{n=1}^{N} w_n \right)^2} \sum_{n=1}^{N} w_n \hat{y}_n \sum_{l=1}^{N} \frac{\partial w_l}{\partial \boldsymbol{x}} \\
&= \frac{1}{\sum_{n=1}^{N} w_n} \sum_{n=1}^{N} \left(-\hat{y}_n w_n \mathbf{D}_n (\boldsymbol{x} - \mathbf{c}_n) + w_n \mathbf{b}_n \right) \\
&\quad - \frac{\hat{f}(\boldsymbol{x})}{\left(\sum_{n=1}^{N} w_n \right)^2} \sum_{n=1}^{N} w_n \mathbf{D}_n (\boldsymbol{x} - \mathbf{c}_n)
\end{aligned}$$

If we learn each output dimension independently, the Jacobian matrix is constructed as:

$$\begin{bmatrix} \dfrac{\partial \hat{\mathbf{f}}}{\partial \boldsymbol{x}} \\ \dfrac{\partial \hat{\mathbf{f}}}{\partial \mathbf{u}} \end{bmatrix} = \frac{\partial}{\partial \boldsymbol{x}} \begin{bmatrix} \hat{f}_1, \ \hat{f}_2, \ \cdots, \ \hat{f}_N \end{bmatrix}^\top$$

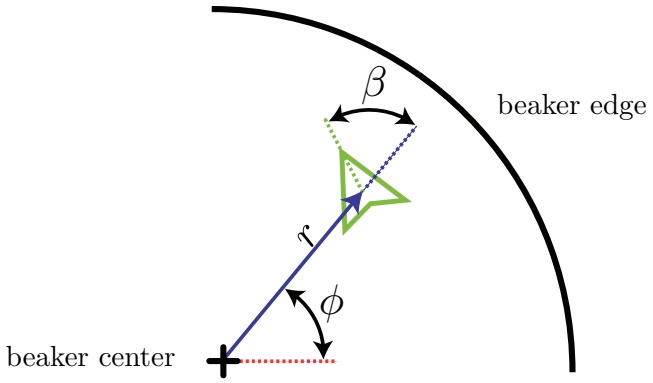

Figure 4.1: Coordinate frame (r, β, ϕ) used for magnetic swimmers. The swimmer is shown in green.

4.3 Swimmer Modeling

The nonlinear dynamical behavior of the swimmer can be described by the difference equation:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}_k(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad k = 0, \ldots, K-1 \tag{4.5}$$

Assuming the system can be approximated to an autonomous model, valid over some time horizon $K-1$, i.e. the duration of the motion primitive, we can rewrite (4.5) as:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k), \quad k = 0, \ldots, K-1 \tag{4.6}$$

Where, for the magnetic swimmer the state and control input are defined as

$$\boldsymbol{x}_k = [r_k,\, \beta_k,\, \phi_k]^\top \quad \boldsymbol{u}_k = u_k \tag{4.7}$$

The state components r_k and ϕ_k are the magnitude and angle of the radial vector which points from the world center to the swimmer; and β_k denotes the heading angle with respect to the radial vector. The Helmholtz coil is energized with a sinusoidal signal $A_k \sin(2\pi\psi_k t)$, where A_k is the amplitude of the signal, ψ_k is the frequency and k is the time index.

The control strategy explained on chapter 5 relies on learning a model for (4.6). To simplify this task we assume swimmer dynamics of the following form:

$$\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \Delta t \cdot \hat{\boldsymbol{f}}(\pi(\boldsymbol{x}_k), \boldsymbol{u}_k) \tag{4.8}$$

$$\begin{array}{ccc} \textit{Inputs} & & \textit{Outputs} \\ \boldsymbol{x} & & \boldsymbol{y} \\ \begin{pmatrix} r_k \\ \beta_k \\ u_k \end{pmatrix} & \begin{array}{c} \hat{\boldsymbol{f}}(\cdot) \\ \Longrightarrow \end{array} & \begin{pmatrix} \hat{r} \\ \hat{\beta} \\ \hat{\phi} \end{pmatrix} \end{array}$$

Figure 4.2: Our model learning problem is defined as obtaining the mapping $\hat{\boldsymbol{f}}$ from input variables $\boldsymbol{x}$ to outputs $\boldsymbol{y}$.

We also assume symmetric dynamics with respect to ϕ, which defines π as

$$\pi : \boldsymbol{x}_k \mapsto r, \beta \tag{4.9}$$

Therefore, the function $\hat{\boldsymbol{f}}$ is learned using LWPR as:

$$\frac{\boldsymbol{x}_{k+1} - \boldsymbol{x}_k}{\Delta t} = \hat{\boldsymbol{f}}(\pi(\boldsymbol{x}_k), \boldsymbol{u}_k) \qquad \pi : \boldsymbol{x}_k \mapsto r, \beta \tag{4.10}$$

Figure 4.2 presents the idea behind learning (4.10).

For the implementation of the forward model and to fit the standard model learning problem of the form in (4.2), $\hat{\mathbf{f}}$ is split into three individual models corresponding to each output variable:

$$\hat{\boldsymbol{f}}(\pi(\boldsymbol{x}_k), \boldsymbol{u}_k) = \begin{bmatrix} M_{\hat{r}}(r, \beta, u) \\ M_{\hat{\beta}}(r, \beta, u) \\ M_{\hat{\phi}}(r, \beta, u) \end{bmatrix} \tag{4.11}$$

and the input-output pairs for each individual model are computed as:

$$\begin{bmatrix} \hat{r} \\ \hat{\beta} \\ \hat{\phi} \end{bmatrix} = \begin{bmatrix} M_{\hat{r}}(r, \beta, u) \\ M_{\hat{\beta}}(r, \beta, u) \\ M_{\hat{\phi}}(r, \beta, u) \end{bmatrix} \tag{4.12}$$

where

$$\begin{aligned} \hat{r} &= (r_{k+1} - r_k)/\Delta t \\ \hat{\beta} &= (\beta_{k+1} - \beta_k)/\Delta t \\ \hat{\phi} &= (\phi_{k+1} - \phi_k)/\Delta t \end{aligned} \tag{4.13}$$

We start the learning process by collecting real-time data. The system is driven for a finite time by a repeating ramp input from 30Hz to 40Hz for 240s, resulting state measurements

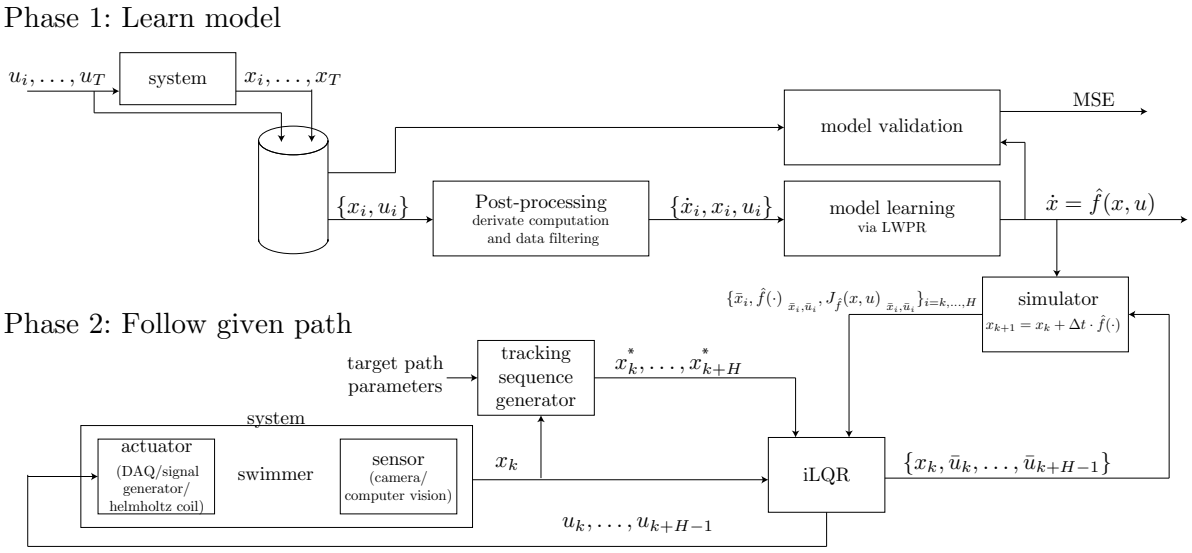

Figure 4.3: Block diagram, explained in Chapters 4 and 5

are stored from the computer vision processing. After more than 20,000 points have been stored, the time history of states is smoothed using a zero-phase filter by passing the data through a low-pass Butterworth filter in both the forward and backward directions. Filtering the data before feeding to LWPR allows us to obtain better estimates of the states by taking advantage of the time dependance. The input-output pairs are computed and fed individually using to LWPR.

The result is not an analytical model of the swimmer, but rather an algorithm that answers a query i.e. a current state, by weighting a series of linear models learned beforehand and returning an approximated state derivative.

As described previously, our objective is to obtain a dynamical model that, given the state $\boldsymbol{x}_k$ and the control input $\boldsymbol{u}_k$ at time $\Delta t \cdot k$, returns an estimate of the next state $\boldsymbol{x}_{k+1}$. The result is a nonlinear estimation for (4.10) which serves as a forward simulator useful for the design of the iterative LQR controller as we will see in Chapter 5.

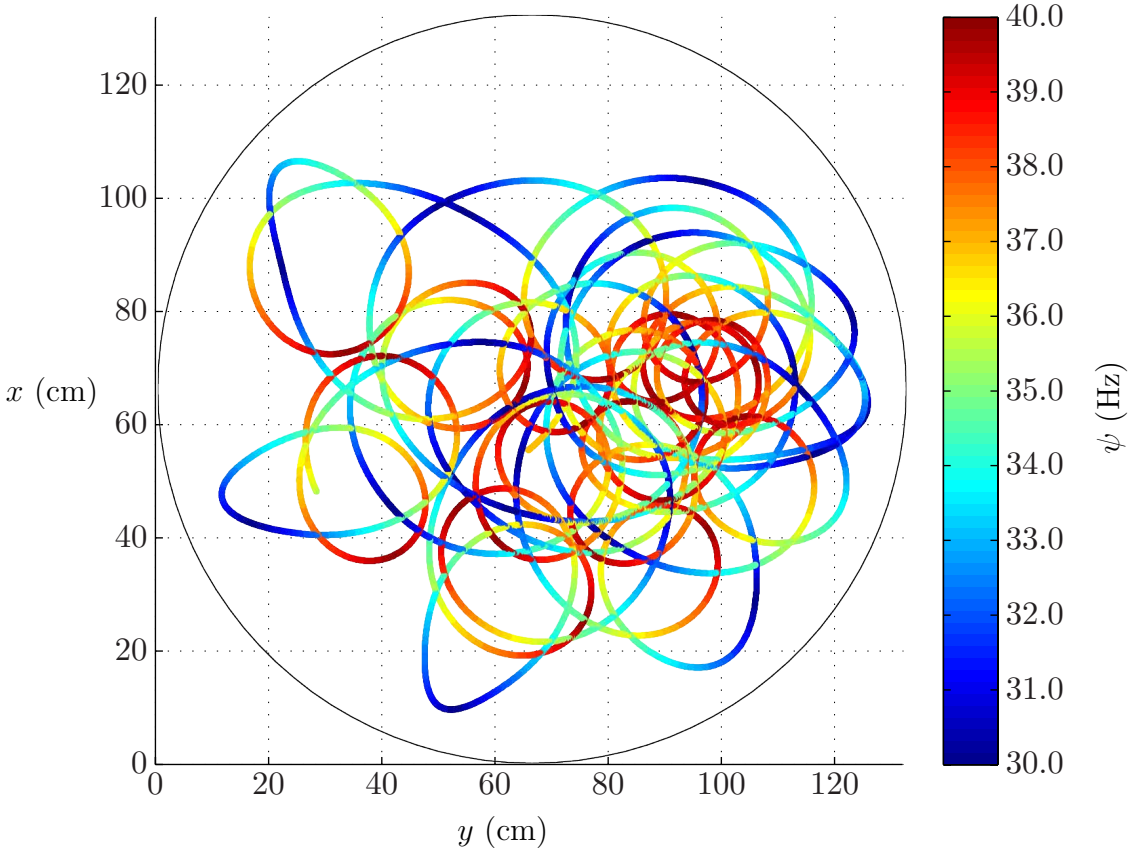

Figure 4.4: Typical swimmer trajectory under a ramp control input. Collected data points are used for learning a model.

CHAPTER 5

CONTROL STRATEGY

Robotic systems often rely on layered architectures to achieve complex tasks. Traditionally four layers have been identified, each with a particular objective:

1. **Task Planning**: the end-user interaction occurs at this level, who provides general characteristics of the task to be achieved, for example initial and final locations. This layer then plans a geometric path that satisfies the given task taking into account environment restrictions.

2. **Trajectory generation**: provides an interpretation of the geometric path in terms of system states taking into account dynamical interactions and system restrictions.

3. **Motion Control**: generates the appropriate inputs that will keep the system within some range of the desired states.

4. **Actuation and Sensing**: provides the interface with the physical environment and returns system state measurements and estimations to the upper layers.

The objective of the control strategy can be defined as to generate a library of trajectories that our robotic swimmer can reliably follow and that will aid for the future task of planning complex movements under environment restrictions. Therefore, we are mainly concerned with the three lowest layers.

Actuation and sensing has been explained on chapter 3. The system dynamics and restrictions have been modeled on chapter 4. This chapter now deals with the computation of a control rule that will select appropriate frequency inputs given a measured state that will keep our swimmer following a circular path.

More specifically, our objective is to obtain a control law $u_k^* = \pi_k(\boldsymbol{x}_k)$ for the task of following circular paths with support of the non-linear dynamical model learned in Chapter 4. By defining our problem on an optimal control setting, we may penalize deviation from the desired state $\boldsymbol{x}_k^*$ and control effort $\boldsymbol{u}_k^*$.

The purpose of the following sections is to present theory in Optimal Control that will allow us to solve:

$$\begin{aligned} \underset{u_k}{\text{minimize}} \quad & \frac{1}{2}\sum_{k=0}^{K}(\delta \boldsymbol{x}_k^\top \boldsymbol{Q}\delta \boldsymbol{x}_k + \delta \boldsymbol{u}_k^\top \boldsymbol{R}\delta \boldsymbol{u}_k) \\ \text{subject to} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \end{aligned} \tag{5.1}$$

where $\delta \boldsymbol{x}_k = \boldsymbol{x}_k - \boldsymbol{x}_k^*, \delta \boldsymbol{u}_k = \boldsymbol{u}_k - \boldsymbol{u}_k^*$ and $\boldsymbol{Q} \geq 0, \boldsymbol{R} > 0$ are the state cost and input cost matrices.

The following chapter proceeds as follows. Section 5.1 presents an overview of the theory for the calculation of a Linear Quadratic Regulator (LQR). Section 5.2 presents a modified version of LQR applicable to non-linear systems. Section 5.3 address particular concerns on the calculation of controller under modeling errors and solves the issue using Model Predictive Control (MPC) and real-time model adaptation. Section 5.4 presents the calculations used to obtain an iLQR controller for the swimmer given the task of trajectory following.

5.1 Linear Quadratic Regulator

Beyond classical control theory, where the primary goal is to stabilize a dynamical system, we might be interested on its operation at a minimum cost, with respect to some given measure of performance. For example, when operating a mobile robot, energy consumption can become a concern. In which case, the controller sending the actuation signal should operate in such a manner that the actuators spend the least amount of energy. In this example the measure of performance or *optimality criterion* would be the energy associated with the control signal.

Optimal Control (OC) sets the mathematical framework for posing and solving this type of control problems. OC is an extensive branch of mathematics and its detailed explanation is beyond the aim of this thesis.

We are mainly concerned with a particular case of OC known as Linear Optimal Control (LOC). In this sort of OC, the plant is assumed to be linear and its dynamics can be described by a set of linear differential equations. A condition is posed on the controller to be linear which is achieved by working with quadratic performance indices. The methods used to solve this case of OC are called Linear Quadratic methods and the solution controller, Linear Quadratic Regulator (LQR) [32].

Framing our problem as shown in (5.7) will aid us in the design of a controller to minimize the deviation from a target trajectory while setting constraints on the characteristics of the control signal. The following subsections present the theory for the solution of LQ problems.

5.1.1 The Regulation Problem

Consider the linear time-invariant discrete-time system described by the linear difference equation:

$$\boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k, \quad k \geq 0$$

The regulator problem is to find control inputs $\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots$ so that states $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ stay close to some desired state, say $\boldsymbol{x}_d$. Assume the desired state is 0.

For $\boldsymbol{x}_d \neq 0$, a change of variable $\boldsymbol{z}_k = \boldsymbol{x}_k - \boldsymbol{x}_d$ and $\boldsymbol{v}_k = \boldsymbol{u}_k - \boldsymbol{u}^*$ (where $\boldsymbol{u}^*$ is an input that keeps the state at $\boldsymbol{x}_d$) will transform the system into the desired form.

The cost function of the regulator problem is defined as:

$$J = \frac{1}{2}\boldsymbol{x}_K^\top \boldsymbol{Q}_f \boldsymbol{x}_K + \frac{1}{2}\sum_{k=0}^{K-1}(\boldsymbol{x}_k^\top \boldsymbol{Q}\boldsymbol{x}_k + \boldsymbol{u}_k^\top \boldsymbol{R}\boldsymbol{u}_k)$$

where $\boldsymbol{Q} = \boldsymbol{Q}^\top \geq 0$, $\boldsymbol{Q}_f = \boldsymbol{Q}_f^\top \geq \boldsymbol{0}$, $\boldsymbol{R} = \boldsymbol{R}^\top > 0$ are the state cost, final cost and input cost matrices. The finite-horizon LQR problem is:

$$\begin{aligned} \underset{\boldsymbol{u}_k}{\text{minimize}} \quad & J = \frac{1}{2}\boldsymbol{x}_K^\top \boldsymbol{Q}_f \boldsymbol{x}_K + \frac{1}{2}\sum_{k=0}^{K-1}(\boldsymbol{x}_k^\top \boldsymbol{Q}\boldsymbol{x}_k + \boldsymbol{u}_k^\top \boldsymbol{R}\boldsymbol{u}_k) \\ \text{subject to} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k, \qquad 0 \leq k \leq K \end{aligned} \tag{5.2}$$

To derive the solution of (5.8) we will use dynamic programming. We start by defining the value function $V_k : \mathbf{R^n} \mapsto \mathbf{R}$ by

$$V_k(\boldsymbol{z}) = \min_{\boldsymbol{u}_k, \ldots, \boldsymbol{u}_{K-1}} J_k$$

subject to $\boldsymbol{x}_k = \boldsymbol{z}$, $\boldsymbol{x}_{j+1} = \boldsymbol{A}\boldsymbol{x}_j + \boldsymbol{B}\boldsymbol{u}_j$, where

$$J_k = \frac{1}{2}\boldsymbol{x}_K^\top \boldsymbol{Q}_f \boldsymbol{x}_K + \frac{1}{2}\sum_{j=K}^{K-1}(\boldsymbol{x}_j^\top \boldsymbol{Q}\boldsymbol{x}_j + \boldsymbol{u}_j^\top \boldsymbol{R}\boldsymbol{u}_j)$$

is a cost-to-go function. Then, the value $V_k(\boldsymbol{z})$ is the minimum cost-to-go, starting from state $\boldsymbol{z}$ at time k and $V_0(\boldsymbol{x}_0)$ is the minimum cost, starting at state $\boldsymbol{x}_0$ at time 0.

A recursive equation is obtained using the principle of optimality and can be written for the optimal value function:

$$V_k(\boldsymbol{z}) = \min_{\boldsymbol{v}} \left(\frac{1}{2}(\boldsymbol{z}^\top \boldsymbol{Q}\boldsymbol{z} + \boldsymbol{v}^\top \boldsymbol{R}\boldsymbol{v}) + V_{k+1}(\boldsymbol{A}\boldsymbol{z} + \boldsymbol{B}\boldsymbol{v}) \right) \qquad 0 \leq k \leq K-1 \tag{5.3}$$

where $\boldsymbol{u}_k = \boldsymbol{v}$ and $\boldsymbol{x}_k = \boldsymbol{z}$. Equation (5.3) is known as the Hamilton-Jacobi-Bellman equation, and provides V_k recursively in terms of V_{k+1}.

Now, assume that $V_N(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^\top \boldsymbol{P}_N \boldsymbol{z}$, from (5.3)

$$V_{N-1}(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^\top \boldsymbol{Q}\boldsymbol{z} + \min_{\boldsymbol{v}} \left(\frac{1}{2}\boldsymbol{v}^\top \boldsymbol{R}\boldsymbol{v} + \frac{1}{2}(\boldsymbol{A}\boldsymbol{z} + \boldsymbol{B}\boldsymbol{v})^\top \boldsymbol{P}_N (\boldsymbol{A}\boldsymbol{z} + \boldsymbol{B}\boldsymbol{v}) \right)$$

Minizing $\boldsymbol{v}$ gives the optimal u_{N-1} and can be obtained from the first order-condition

$$\frac{\partial V_{N-1}}{\partial \boldsymbol{v}} = 0$$

which gives

$$0 = (\boldsymbol{R} + \boldsymbol{B}^\top \boldsymbol{P}_N \boldsymbol{B})\boldsymbol{v} + \boldsymbol{B}^\top \boldsymbol{P}_N \boldsymbol{A}\boldsymbol{z}$$

We can conclude there is only one stationary point and therefore, the global minimum

$$v^* = -(\boldsymbol{R} + \boldsymbol{B}^\top \boldsymbol{P}_N \boldsymbol{B})^{-1} \boldsymbol{B}^\top \boldsymbol{P}_N \boldsymbol{A}\boldsymbol{z} \tag{5.4}$$

since $\boldsymbol{P}_N = \boldsymbol{Q}_f > 0$, $\boldsymbol{R} \geq 0$.

Substituting back to the recursive formula gives

$$\begin{aligned} V_{N-1}(\boldsymbol{z}) &= \frac{1}{2}\boldsymbol{z}^\top \boldsymbol{Q}\boldsymbol{z} + \frac{1}{2}\boldsymbol{v}^{*\top} \boldsymbol{R}\boldsymbol{v}^* + \frac{1}{2}(\boldsymbol{A}\boldsymbol{z} + \boldsymbol{B}\boldsymbol{v}^*)^\top \boldsymbol{P}_N (\boldsymbol{A}\boldsymbol{z} + \boldsymbol{B}\boldsymbol{v}^*) \\ &= \frac{1}{2}\boldsymbol{z}^\top \left(\boldsymbol{Q} + \boldsymbol{A}^\top \boldsymbol{P}_N \boldsymbol{A} - \boldsymbol{A}^\top \boldsymbol{P}_N \boldsymbol{B}(\boldsymbol{R} + \boldsymbol{B}^\top \boldsymbol{P}_N \boldsymbol{B})^{-1} \boldsymbol{B}^\top \boldsymbol{P}_N \boldsymbol{A} \right) \boldsymbol{z} \end{aligned}$$

We can identify a quadratic recursive expression

$$V_{N-1}(\boldsymbol{z}) = \frac{1}{2}\boldsymbol{z}^\top \boldsymbol{P}_{N-1} \boldsymbol{z}$$

where

$$\boldsymbol{P}_{N-1} = \boldsymbol{Q} + \boldsymbol{A}^\top \boldsymbol{P}_N \boldsymbol{A} - \boldsymbol{A}^\top \boldsymbol{P}_N \boldsymbol{B}(\boldsymbol{R} + \boldsymbol{B}^\top \boldsymbol{P}_N \boldsymbol{B})^{-1} \boldsymbol{B}^\top \boldsymbol{P}_N \boldsymbol{A}$$

Hence the optimal control, given $\boldsymbol{x}_{N-1} = \boldsymbol{z}$, is

$$\boldsymbol{u}_{N-1} = -(\boldsymbol{R} + \boldsymbol{B}^\top \boldsymbol{P}_N \boldsymbol{B})^{-1} \boldsymbol{B}^\top \boldsymbol{P}_N \boldsymbol{A}\boldsymbol{x}_{N-1}$$

and the recursion continues for all k and the summarized algorithm for obtaining the optimal $\boldsymbol{u}_k$ for the optimal control problem (5.8) is outlined on the algorithm presented on figure 5.1.

1. set $\boldsymbol{P}_K := \boldsymbol{Q}_f$
2. for $k = K, \ldots, 1$ do
   $\boldsymbol{P}_{k-1} := \boldsymbol{Q} + \boldsymbol{A}^\top \boldsymbol{P}_k \boldsymbol{A} - \boldsymbol{A}^\top \boldsymbol{P}_k \boldsymbol{B}(\boldsymbol{R} + \boldsymbol{B}^\top \boldsymbol{P}_k \boldsymbol{B})^{-1} \boldsymbol{B}^\top \boldsymbol{P}_k \boldsymbol{A}$
3. for $k = 0, \ldots, K-1$ do
   $\boldsymbol{K}_k := -(\boldsymbol{R} + \boldsymbol{B}^\top \boldsymbol{P}_{k+1} \boldsymbol{B})^{-1} \boldsymbol{B}^\top \boldsymbol{P}_{k+1} \boldsymbol{A}$
4. for $k = 0, \ldots, K-1$ do
   $\boldsymbol{u}_k^{LQR} = \boldsymbol{K}_k \boldsymbol{x}_k$

Figure 5.1: Solution algorithm to the LQR problem

1. set $\boldsymbol{P}_K := \boldsymbol{Q}_f$
2. for $k = K, \ldots, 1$ do
   $\boldsymbol{P}_{k-1} := \boldsymbol{Q}_{k-1} + \boldsymbol{A}_{k-1}^\top \boldsymbol{P}_k \boldsymbol{A}_{k-1} - \boldsymbol{A}_{k-1}^\top \boldsymbol{P}_k \boldsymbol{B}_{k-1}(\boldsymbol{R}_{k-1} + \boldsymbol{B}_{k-1}^\top \boldsymbol{P}_k \boldsymbol{B}_{k-1})^{-1} \boldsymbol{B}_{k-1}^\top \boldsymbol{P}_k \boldsymbol{A}_{k-1}$
3. for $k = 0, \ldots, K-1$ do
   $\boldsymbol{K}_k := -(\boldsymbol{R}_k + \boldsymbol{B}_k^\top \boldsymbol{P}_{k+1} \boldsymbol{B}_k)^{-1} \boldsymbol{B}_k^\top \boldsymbol{P}_{k+1} \boldsymbol{A}_k$
4. for $k = 0, \ldots, K-1$ do
   $\boldsymbol{u}_k^{LQR} = \boldsymbol{K}_k \boldsymbol{x}_k$

Figure 5.2: Solution algorithm to the time-varying LQR problem

For a linear time-varying system and a quadratic time-varying cost function, the finite-horizon LQR problem is:

$$\begin{aligned} \underset{\boldsymbol{u}_k}{\text{minimize}} \quad & J = \frac{1}{2}\boldsymbol{x}_K^\top \boldsymbol{Q}_f \boldsymbol{x}_K + \frac{1}{2}\sum_{k=0}^{K-1}(\boldsymbol{x}_k^\top \boldsymbol{Q}_k \boldsymbol{x}_k + \boldsymbol{u}_k^\top \boldsymbol{R}_k \boldsymbol{u}_k) \\ \text{subject to} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{A}_k \boldsymbol{x}_k + \boldsymbol{B}_k \boldsymbol{u}_k \qquad\qquad k \geq 0 \end{aligned} \tag{5.5}$$

The dynamic programming approach used before for finding the solution to the LQR problem is readily extended to time-varying systems. The summarized algorithm for obtaining the optimal $\boldsymbol{u}_k$ for the optimal control problem (5.5) is outlined on the algorithm presented on figure 5.2.

5.1.2 The Trajectory Tracking Problem

In a tracking problem, it is desired to have the state close to a nominal feasible trajectory. Denote by $\bar{\boldsymbol{x}}_k$ and $\bar{\boldsymbol{u}}_k$ the nominal state and input trajectories respectively. We can penalize the deviation from the trajectory:

$$\delta \boldsymbol{x}_k = \boldsymbol{x}_k - \bar{\boldsymbol{x}}_k$$
$$\delta \boldsymbol{u}_k = \boldsymbol{u}_k - \bar{\boldsymbol{u}}_k$$

and express the problem in terms of these deviations

$$\begin{aligned} \underset{\boldsymbol{u}_k}{\text{minimize}} \quad & J = \frac{1}{2}\delta \boldsymbol{x}_K^\top \boldsymbol{Q}_f \delta \boldsymbol{x}_K + \frac{1}{2}\sum_{k=0}^{K-1} \delta \boldsymbol{x}_k^\top \boldsymbol{Q} \delta \boldsymbol{x}_k + \delta \boldsymbol{u}_k^\top \boldsymbol{R} \delta \boldsymbol{u}_k \\ \text{subject to} \quad & \delta \boldsymbol{x}_{k+1} = \boldsymbol{A}_k \delta \boldsymbol{x}_k + \boldsymbol{B}_k \delta \boldsymbol{u}_k \qquad k \geq 0 \end{aligned} \tag{5.6}$$

where $\boldsymbol{Q} = \boldsymbol{Q}^\top \geq 0$, $\boldsymbol{Q}_f = \boldsymbol{Q}_f^\top \geq \boldsymbol{0}$, $\boldsymbol{R} = \boldsymbol{R}^\top > 0$ are the state cost, final cost and input cost matrices.

Yielding a finite-horizon and time-varying LQR, which is on the form of (5.5) and the solution is given by algorithm 5.2. The optimal control input is then:

$$\underbrace{\delta \boldsymbol{u}_k^{LQR}}_{\boldsymbol{u}_k^{LQR} - \bar{\boldsymbol{u}}_k} = \boldsymbol{K}_k \underbrace{\delta \boldsymbol{x}_k}_{\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k}$$

or

$$\boldsymbol{u}_k^{LQR} = \boldsymbol{K}_k(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \bar{\boldsymbol{u}}_k$$

5.1.3 Regulation on Non-linear Systems

From the problem stated in (5.7), we have developed solutions for the case where $\boldsymbol{f}(\cdot)$ was linear and J quadratic. We will next explore situations where $\boldsymbol{f}(\cdot)$ is allowed to be non-linear.

Consider

$$\begin{aligned} \underset{\boldsymbol{u}_k}{\text{minimize}} \quad & J = \frac{1}{2}\boldsymbol{x}_K^\top \boldsymbol{Q}_f \boldsymbol{x}_K + \frac{1}{2}\sum_{k=0}^{K-1} (\boldsymbol{x}_k^\top \boldsymbol{Q} \boldsymbol{x}_k + \boldsymbol{u}_k^\top \boldsymbol{R} \boldsymbol{u}_k) \\ \text{subject to} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \qquad k \geq 0 \end{aligned}$$

This is a regulation problem i.e. we seek control inputs $\boldsymbol{u}_0, \boldsymbol{u}_1, \ldots$ so that states $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$ stay close to some desired state, say $\boldsymbol{x}_d = \bar{\boldsymbol{x}}$. The optimal control problem is to find the

control sequence $\{\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{K-1}\}$ so that the cost function J is minimized. We set a condition to the desired state to be an equilibrium point:

$$\exists \bar{\boldsymbol{u}} : \quad \bar{\boldsymbol{x}} = \boldsymbol{f}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})$$

Assuming the system will stay close to $\bar{\boldsymbol{x}}$, we linearize the dynamics about it:

$$\boldsymbol{x}_{k+1} \approx \overbrace{\boldsymbol{f}(\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}})}^{\bar{\boldsymbol{x}}} + \underbrace{\left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}}}_{\boldsymbol{A}} (\boldsymbol{x}_k - \bar{\boldsymbol{x}}) + \underbrace{\left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_{\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}}}_{\boldsymbol{B}} (\boldsymbol{u}_k - \bar{\boldsymbol{u}})$$

let

$$\delta \boldsymbol{x}_k = \boldsymbol{x}_k - \bar{\boldsymbol{x}}$$
$$\delta \boldsymbol{u}_k = \boldsymbol{u}_k - \bar{\boldsymbol{u}}$$

then, we get the following problem

$$\begin{aligned} \underset{u_k}{\text{minimize}} \quad & J = \frac{1}{2} \delta \boldsymbol{x}_K^\top \boldsymbol{Q}_f \delta \boldsymbol{x}_K + \frac{1}{2} \sum_{k=0}^{K-1} \delta \boldsymbol{x}_k^\top \boldsymbol{Q} \delta \boldsymbol{x}_k + \delta \boldsymbol{u}_k^\top \boldsymbol{R} \delta \boldsymbol{u}_k \\ \text{subject to} \quad & \delta \boldsymbol{x}_{k+1} = \boldsymbol{A} \delta \boldsymbol{x}_k + \boldsymbol{B} \delta \boldsymbol{u}_k \qquad k \geq 0 \end{aligned}$$

This problem is of the form in (5.5) and the solution is given by algorithm 5.2.

5.1.4 Trajectory Tracking on Non-linear Systems with Feasible Target States

It is desired to have the state close to a nominal sequence $\bar{\boldsymbol{x}}_0, \bar{\boldsymbol{x}}_1, \ldots, \bar{\boldsymbol{x}}_K$. The sequence is feasible if and only if

$$\exists \bar{\boldsymbol{u}}_0, \bar{\boldsymbol{u}}_1, \ldots, \bar{\boldsymbol{u}}_{K-1} : \quad \forall t \in \{0, 1, \ldots, K-1\} : \quad \bar{\boldsymbol{x}}_{k+1} = \boldsymbol{f}(\bar{\boldsymbol{x}}_{k+1}, \boldsymbol{u}_{k+1})$$

We can penalize the deviation from the trajectory:

$$\delta \boldsymbol{x}_k = \boldsymbol{x}_k - \bar{\boldsymbol{x}}_k$$
$$\delta \boldsymbol{u}_k = \boldsymbol{u}_k - \bar{\boldsymbol{u}}_k$$

Then the problem statement is

$$\begin{aligned} \underset{u_k}{\text{minimize}} \quad & J = \frac{1}{2}\delta \boldsymbol{x}_K^\top \boldsymbol{Q}_f \delta \boldsymbol{x}_K + \frac{1}{2}\sum_{k=0}^{K-1}(\delta \boldsymbol{x}_k^\top \boldsymbol{Q}\delta \boldsymbol{x}_k + \delta \boldsymbol{u}_k^\top \boldsymbol{R}\delta \boldsymbol{u}_k) \\ \text{subject to} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \qquad\qquad k \geq 0 \end{aligned}$$

Assuming the system will stay close to $\bar{\boldsymbol{x}}_k$ we linearize the dynamics about it:

$$\boldsymbol{x}_{k+1} \approx \overbrace{\boldsymbol{f}(\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k)}^{\bar{\boldsymbol{x}}_{k+1}} + \underbrace{\left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k}}_{\boldsymbol{A}_k}(\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \underbrace{\left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_{\bar{\boldsymbol{x}}_k, \bar{\boldsymbol{u}}_k}}_{\boldsymbol{B}_k}(\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k)$$

by changing variables to $\delta \boldsymbol{x}_k$, and $\delta \boldsymbol{u}_k$ we get the following problem

$$\begin{aligned} \underset{u_k}{\text{minimize}} \quad & J = \frac{1}{2}\delta \boldsymbol{x}_K^\top \boldsymbol{Q}_f \delta \boldsymbol{x}_K + \frac{1}{2}\sum_{k=0}^{K-1}\delta \boldsymbol{x}_k^\top \boldsymbol{Q}\delta \boldsymbol{x}_k + \delta \boldsymbol{u}_k^\top \boldsymbol{R}\delta \boldsymbol{u}_k \\ \text{subject to} \quad & \delta \boldsymbol{x}_{k+1} = \boldsymbol{A}_k \delta \boldsymbol{x}_k + \boldsymbol{B}_k \delta \boldsymbol{u}_k \qquad\qquad k \geq 0 \end{aligned}$$

This problem is of the form in (5.5) and the solution is given by algorithm 5.2.

By linearizing we are assuming the error will stay close to the desired feasible trajectory, and by doing a change of variables to describe the error, we are moving our coordinate axis along the given trajectory. These substitutions allow to approximate the problem to a linear time-varying system, which can be controlled through an LQR scheme. Therefore, within some range, the controller is capable to return the system to the desired trajectory in the presence of perturbations and initial conditions being off, even with model dynamics being off.

5.2 Iterative LQR

Iterative LQR (iLQR) [7], also known as Gauss-Newton LQR and Sequential LQR, is a method for solving optimal control problems with non-linear cost functions and dynamical model:

$$\begin{aligned} \underset{\boldsymbol{u}_k}{\text{minimize}} \quad & J_k = \sum_{k=0}^{K} \boldsymbol{l}_k(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ \text{subject to} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \end{aligned} \tag{5.7}$$

We now allow $\boldsymbol{f}(\cdot)$ and $\boldsymbol{l}_k(\cdot)$ to be non-linear and $\boldsymbol{x}_k^*$.

The method begins with an initial control guess $\bar{\boldsymbol{U}}_k^{(0)} = \{\bar{\boldsymbol{u}}_0^{(0)}, \bar{\boldsymbol{u}}_1^{(0)}, \ldots, \bar{\boldsymbol{u}}_K^{(0)}\}$, and the corresponding nominal state sequence $\bar{\boldsymbol{X}}_k^{(0)} = \{\bar{\boldsymbol{x}}_0^{(0)}, \bar{\boldsymbol{x}}_1^{(0)}, \ldots, \bar{\boldsymbol{x}}_K^{(0)}\}$ obtained from (5.7), and continues by computing a linear approximation of the dynamics and a quadratic one for the cost around the nominal trajectory $\bar{\boldsymbol{X}}_k^{(i)}$.

Essentially iLQR transforms our non-linear optimal control problem to a linear time-varying one.

$$\begin{aligned} \underset{\boldsymbol{u}_k}{\text{minimize}} \quad & J = \sum_{k=0}^{K} (\boldsymbol{x}_k^\top \boldsymbol{Q} \boldsymbol{x}_k + \boldsymbol{u}_k^\top \boldsymbol{R} \boldsymbol{u}_k) \\ \text{subject to} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{A}\boldsymbol{x}_k + \boldsymbol{B}\boldsymbol{u}_k \end{aligned} \tag{5.8}$$

Solving the LQ formulation will provide an improved control sequence $\bar{\boldsymbol{U}}_k^{(i)}$ and corresponding state trajectory $\bar{\boldsymbol{X}}_k^{(i)}$. We iterate over the last state trajectory or exit if $J^{(i)} < tol \cdot J^{(i-1)}$. It is relevant to highlight that $\boldsymbol{x}_k^*$, $\boldsymbol{u}_k^*$ need not be feasible for this method to work.

In solving control problem of the LQ approximation, the cost function rewards the system for driving towards the minimum of the cost function. Since the linearized model is valid near $\boldsymbol{x}_k^0$, the resulting controller on the current iteration might not take the current state trajectory close to the desired one. The algorithm proceeds by replacing $\boldsymbol{u}_k^0$ with the new LQR feedback policy and control sequence $\boldsymbol{u}_k^1$ and computes the corresponding new states $\boldsymbol{x}_k^1$. This new controller will perform better than the previous on getting the states close to the desired trajectory. Since this is a non-convex optimization problem, there are no guarantees of convergence to the globally optimal $\boldsymbol{U}^*$.

We linearize the system dynamics in step 2.1 on the iLQR algorithm described on 5.3 as follows.

Take the Taylor series expansion up to the 1st order terms around the nominal trajectory on the current iteration ($\bar{\boldsymbol{x}}_k{}^{(i)}$, $\bar{\boldsymbol{u}}_k{}^{(i)}$)

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\bar{\boldsymbol{x}}_k^{(i)}, \bar{\boldsymbol{u}}_k^{(i)}) + \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\right|_{\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{u}}^{(i)}} (\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k^{(i)}) + \left.\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\right|_{\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{u}}^{(i)}} (\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k^{(i)}) \tag{5.9}$$

add and subtract the target trajectory and control input to express the dynamics in terms

0 Set $i = 0$. Define an initial time-discreted control sequence $\bar{\boldsymbol{U}}_k^{(i)} = \{\bar{\boldsymbol{u}}_0^{(i)}, \bar{\boldsymbol{u}}_1^{(i)}, \ldots, \bar{\boldsymbol{u}}_K^{(i)}\}$, which can be chosen arbitrarily.

1 Obtain the state trajectory $\bar{\boldsymbol{X}}_k^{(i)} = \{\bar{\boldsymbol{x}}_0^{(i)}, \bar{\boldsymbol{x}}_1^{(i)}, \ldots, \bar{\boldsymbol{x}}_K^{(i)}\}$ corresponding to $\bar{\boldsymbol{U}}_k^{(i)}$ by simulation on the forward dynamics model:

$$\boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k)$$

2 Compute the LQ approximation of the optimal control problem around $\bar{\boldsymbol{X}}_k$ and $\bar{\boldsymbol{U}}_k$, i.e.

1. Compute first-order Taylor expansion of the dynamics to obtain:

$$\boldsymbol{z}_{k+1} = \boldsymbol{A}_k \boldsymbol{z}_k + \boldsymbol{B}_k \boldsymbol{v}_k$$

2. Compute second-order Taylor expansion of the cost to obtain:

$$J^{(i)} = \sum_{k=0}^{T} \boldsymbol{z}^\top \boldsymbol{Q}_k \boldsymbol{z} + \boldsymbol{v}^\top \boldsymbol{R}_k \boldsymbol{v}$$

3 Solve the LQ problem, using the algorithm on figure 5.2.

4 Update the control sequence

$$\pi^{i+1} : \ \boldsymbol{u}_k^{(i+1)} = \boldsymbol{u}_k^{(i)} + \boldsymbol{v}_k$$

5 If $J^{(i)} < tol \cdot J^{(i-1)}$ Exit with $\boldsymbol{U}_k^* = \boldsymbol{U}_k^{(i+1)}$
else Set $i = i + 1$ and go to Step 1

Figure 5.3: iterative LQR (iLQR) algorithm

of the tracking error

$$\begin{aligned}\boldsymbol{x}_{k+1} - \boldsymbol{x}^*_{k+1} &= \boldsymbol{f}(\bar{\boldsymbol{x}}^{(i)}_k, \bar{\boldsymbol{u}}^{(i)}_k) - \boldsymbol{x}^*_{k+1} \\ &+ \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\bigg|_{\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{u}}^{(i)}} (\boldsymbol{x}_k - \bar{\boldsymbol{x}}^{(i)}_k - \boldsymbol{x}^*_k + \boldsymbol{x}^*_k) \\ &+ \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\bigg|_{\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{u}}^{(i)}} (\boldsymbol{u}_k - \bar{\boldsymbol{u}}^{(i)}_k - \boldsymbol{u}^*_k + \boldsymbol{u}^*_k)\end{aligned}$$

and by expansion of the terms, we obtain

$$\begin{aligned}\boldsymbol{x}_{k+1} - \boldsymbol{x}^*_{k+1} &= \boldsymbol{f}(\bar{\boldsymbol{x}}^{(i)}_k, \bar{\boldsymbol{u}}^{(i)}_k) - \boldsymbol{x}^*_{k+1} \\ &+ \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\bigg|_{\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{u}}^{(i)}} (\boldsymbol{x}_k - \boldsymbol{x}^*_k) + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\bigg|_{\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{u}}^{(i)}} (\boldsymbol{u}_k - \boldsymbol{u}^*_k) \\ &+ \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\bigg|_{\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{u}}^{(i)}} (\boldsymbol{x}^*_k - \bar{\boldsymbol{x}}^{(i)}_k) + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\bigg|_{\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{u}}^{(i)}} (\boldsymbol{u}^*_k - \bar{\boldsymbol{u}}^{(i)}_k)\end{aligned}$$

and finally, the linearized dynamics in terms of the errors ($\boldsymbol{z}_k$, $\boldsymbol{u}_k$)

$$\underbrace{\begin{bmatrix} \boldsymbol{x}_{k+1} - \boldsymbol{x}^*_{k+1} \\ 1 \end{bmatrix}}_{\boldsymbol{z}_{k+1}} = \boldsymbol{A}_k \underbrace{\begin{bmatrix} \boldsymbol{x}_k - \boldsymbol{x}^*_k \\ 1 \end{bmatrix}}_{\boldsymbol{z}_k} + \boldsymbol{B}_k \underbrace{(\boldsymbol{u}_k - \boldsymbol{u}^*_k)}_{\boldsymbol{v}_k}$$

where

$$\boldsymbol{A}_k = \begin{bmatrix} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\Big|_{\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{u}}^{(i)}} & \boldsymbol{f}(\bar{\boldsymbol{x}}^{(i)}_k, \bar{\boldsymbol{u}}^{(i)}_k) - \boldsymbol{x}^*_{k+1} + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{x}}\Big|_{\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{u}}^{(i)}} (\boldsymbol{x}^*_k - \bar{\boldsymbol{x}}^{(i)}_k) + \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\Big|_{\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{u}}^{(i)}} (\boldsymbol{u}^*_k - \bar{\boldsymbol{u}}^{(i)}_k) \\ 0 & 1 \end{bmatrix}$$

and

$$\boldsymbol{B}_k = \begin{bmatrix} \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{u}}\Big|_{\bar{\boldsymbol{x}}^{(i)}, \bar{\boldsymbol{u}}^{(i)}} \\ 0 \end{bmatrix}$$

5.3 Tracking Motion Primitives

As described before, our objective is to obtain a control law $u_k^* = \pi_k(\boldsymbol{x}_k)$ for the task of tracking circular paths. Our approach is to define the problem on an optimal control setting:

$$\begin{aligned} \underset{u_k}{\text{minimize}} \quad & \frac{1}{2}\sum_{k=0}^{K}(\delta\boldsymbol{x}_k^\top \boldsymbol{Q}\delta\boldsymbol{x}_k + \delta\boldsymbol{u}_k^\top \boldsymbol{R}\delta\boldsymbol{u}_k) \\ \text{subject to} \quad & \boldsymbol{x}_{k+1} = \boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ & \qquad = \boldsymbol{x}_k + \Delta t \cdot \hat{\boldsymbol{f}}(\pi(\boldsymbol{x}_k), \boldsymbol{u}_k) \end{aligned} \tag{5.10}$$

where

$$\boldsymbol{x}_k = [r_k,\, \beta_k,\, \phi_k]^\top \quad \boldsymbol{u}_k = u_k$$

and $\hat{\boldsymbol{f}}(\cdot)$ is the non-linear dynamical model as described in Chapter 4. To solve (5.10), we apply the method described in section 5.2 with extensions to improve the quality of our controller. These extensions are described over the next subsections.

5.3.1 Shifting Dynamics to Improve Convergence

iLQR as applied on figure 5.3 need not converge, since the optimal policy for the LQ approximation might end up not staying close to the sequence of points around which the LQ approximation was computed by Taylor expansion.

To improve convergence to local optimum we gradually shift the dynamics between iterations from being close to the desired trajectory to the actual system dynamics:

$$\begin{aligned} \boldsymbol{x}_{k+1} &= \beta\boldsymbol{x}_k^* + (1-\beta)\boldsymbol{f}(\boldsymbol{x}_k, \boldsymbol{u}_k) \\ &= \beta\boldsymbol{x}_k^* + (1-\beta)(\boldsymbol{x}_k + \Delta t \cdot \hat{\boldsymbol{f}}(\pi(\boldsymbol{x}_k), \boldsymbol{u}_k)) \end{aligned}$$

We derive the linearized dynamics used in step 2.1 on the iLQR algorithm 5.3 on the next steps.

Given

$$\alpha = 1 - \beta$$

our shifted system dynamics are

$$\boldsymbol{x}_{k+1} = \beta \boldsymbol{x}_k^* + \alpha(\boldsymbol{x}_k + \Delta t \cdot \hat{\boldsymbol{f}}(\pi(\boldsymbol{x}_k), \boldsymbol{u}_k))$$

substituting our LWPR model $\hat{\boldsymbol{f}}(\cdot)$ by taking its Taylor series expansion up to the 1st order terms around our nominal trajectory ($\bar{\boldsymbol{x}}_k$, $\bar{\boldsymbol{u}}_k$)

$$\boldsymbol{x}_{k+1} = \beta \boldsymbol{x}_k^* + \alpha \boldsymbol{x}_k + \alpha \Delta t \hat{\boldsymbol{f}}(\pi(\bar{\boldsymbol{x}}_k), \bar{\boldsymbol{u}_k}) + \alpha \Delta t \frac{\partial \hat{\boldsymbol{f}}}{\partial \boldsymbol{x}}\bigg|_{\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}} (\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k) + \alpha \Delta \frac{\partial \hat{\boldsymbol{f}}}{\partial \boldsymbol{u}}\bigg|_{\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}} (\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k)$$

adding and subtracting $\boldsymbol{x}_k^*$, $\boldsymbol{u}_k^*$ to express the dynamics in terms of the deviation from the target trajectory

$$\begin{aligned} \boldsymbol{x}_{k+1} - \boldsymbol{x}_{k+1}^* &= \beta \boldsymbol{x}_k^* + \alpha \boldsymbol{x}_k - \alpha \boldsymbol{x}_k^* + \alpha \boldsymbol{x}_k^* - \boldsymbol{x}_{k+1}^* \\ &\quad + \alpha \Delta t \hat{\boldsymbol{f}}(\pi(\bar{\boldsymbol{x}}_k), \bar{\boldsymbol{u}_k}) \\ &\quad + \alpha \Delta t \frac{\partial \hat{\boldsymbol{f}}}{\partial \boldsymbol{x}}\bigg|_{\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}} (\boldsymbol{x}_k - \bar{\boldsymbol{x}}_k - \boldsymbol{x}_k^* + \boldsymbol{x}_k^*) + \alpha \Delta t \frac{\partial \hat{\boldsymbol{f}}}{\partial \boldsymbol{u}}\bigg|_{\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}} (\boldsymbol{u}_k - \bar{\boldsymbol{u}}_k - \boldsymbol{u}_k^* + \boldsymbol{u}_k^*) \end{aligned}$$

which we can express as a linear time-varying system

$$\boldsymbol{z}_{k+1} = \boldsymbol{A}_k \boldsymbol{z}_k + \boldsymbol{B}_k \boldsymbol{v}_k \tag{5.11}$$

where

$$\boldsymbol{z}_k = \begin{bmatrix} \boldsymbol{x}_k - \boldsymbol{x}_k^* \\ 1 \end{bmatrix} \qquad \boldsymbol{v}_k = \boldsymbol{u}_k - \boldsymbol{u}_k^* \tag{5.12}$$

and

$$\boldsymbol{A}_k = \begin{bmatrix} \boldsymbol{N}_k & \boldsymbol{M}_k \\ 0 & 1 \end{bmatrix} \qquad \boldsymbol{B}_k = \begin{bmatrix} \alpha \Delta t \frac{\partial \hat{\boldsymbol{f}}}{\partial \boldsymbol{u}}\Big|_{\bar{\boldsymbol{x}}, \bar{\boldsymbol{u}}} \\ 0 \end{bmatrix} \tag{5.13}$$

where

$$
\begin{aligned}
\boldsymbol{N}_k =& \alpha I + \alpha \Delta t \frac{\partial \hat{\boldsymbol{f}}}{\partial \boldsymbol{x}}\bigg|_{\bar{\boldsymbol{x}},\bar{\boldsymbol{u}}} \\
\boldsymbol{M}_k =& \beta \boldsymbol{x}_k^* + \alpha \boldsymbol{x}_k^* - \boldsymbol{x}_{k+1}^* + \alpha \Delta t \hat{\boldsymbol{f}}(\pi(\bar{\boldsymbol{x}}_k), \bar{\boldsymbol{u}}_k) \\
&+ \alpha \Delta t \frac{\partial \hat{\boldsymbol{f}}}{\partial \boldsymbol{x}}\bigg|_{\bar{\boldsymbol{x}},\bar{\boldsymbol{u}}} (\boldsymbol{x}_k^* - \bar{\boldsymbol{x}}_k) + \alpha \Delta t \frac{\partial \hat{\boldsymbol{f}}}{\partial \boldsymbol{u}}\bigg|_{\bar{\boldsymbol{x}},\bar{\boldsymbol{u}}} (\boldsymbol{u}_k^* - \bar{\boldsymbol{u}}_k)
\end{aligned}
\tag{5.14}
$$

In practice, computational speed restricts the number of iterations we can do during realtime control, but experimentally, three iterations with $\beta = 0.7, 0.1, 0.05$ suffices for convergence.

5.3.2 Penalizing the Change in Control Input for Smoothness

The LQ approximation of the dynamics and cost are expressed in terms of $\boldsymbol{z}_k$, $\boldsymbol{v}_k$ driving the tracking error to zero on (5.11) to (5.14). However, in practice, the control signal contains high frequency components. Here we extend the dynamics in order to penalize an immediate change in control input

$$
\begin{aligned}
\underset{\bar{v}_k}{\text{minimize}} \quad & \bar{J} = \frac{1}{2} \sum_{k=0}^{K} \bar{\boldsymbol{z}}_k^\top \bar{\boldsymbol{Q}} \bar{\boldsymbol{z}}_k + \bar{\boldsymbol{v}}_k^\top \bar{\boldsymbol{R}} \bar{\boldsymbol{v}}_k \\
\text{subject to} \quad & \bar{\boldsymbol{z}}_{k+1} = \bar{\boldsymbol{A}}_k \bar{\boldsymbol{z}}_k + \bar{\boldsymbol{B}}_k \bar{\boldsymbol{v}}_k
\end{aligned}
\tag{5.15}
$$

where

$$
\begin{aligned}
&\bar{\boldsymbol{z}}_k = [\boldsymbol{z}_k,\, \boldsymbol{v}_{k-1}]^\top \qquad \bar{\boldsymbol{v}}_k = \boldsymbol{v}_k - \boldsymbol{v}_{k-1} \\
&\bar{\boldsymbol{A}}_k = \begin{bmatrix} \boldsymbol{A}_k & \boldsymbol{B}_k \\ 0 & I \end{bmatrix} \qquad \bar{\boldsymbol{B}}_k = \begin{bmatrix} \boldsymbol{B}_k \\ I \end{bmatrix}
\end{aligned}
\tag{5.16}
$$

This extension allows to set a cost for the difference $\boldsymbol{v}_k - \boldsymbol{v}_{k-1}$ and adjust the smoothness of our control signal. Our new cost matrices are

$$
\bar{\boldsymbol{Q}} = \begin{bmatrix} \boldsymbol{Q} & 0 \\ 0 & \boldsymbol{R} \end{bmatrix}
$$

$$
\bar{\boldsymbol{R}} = [\text{penalty for change in control}]
$$

5.3.3 Obtaining the Initial Guess Based on Curvature Data

To characterize the swimmer we take velocity and curvature measurements, estimated from a history of swimmer locations obtained through the computer vision process. Velocity is directly approximated using finite differences:

$$\boldsymbol{v}_k = \sqrt{\left(\frac{x_k - x_{k-1}}{\Delta t}\right)^2 + \left(\frac{y_k - y_{k-1}}{\Delta t}\right)^2} \tag{5.17}$$

where $\boldsymbol{v}_k$ is the velocity is the forward velocity and (x_k,y_k) the location of the swimmer's head in Cartesian coordinates at time $k \cdot \Delta t$.

To quantify the steepness of a turn achieved by a swimmer at a particular frequency we use an extrinsic curvature on its simplest form, also known as first curvature (see [11]). A curve embedded on a 2-D Euclidean space given by Cartesian parametric equations $x = x(t)$ and $y = y(t)$ has the following first curvature

$$\kappa = \frac{d\theta}{ds} \tag{5.18}$$

$$= \frac{\frac{d\theta}{dt}}{\sqrt{\left(\frac{dx}{dt}\right)^2 + \left(\frac{dy}{dt}\right)^2}} \tag{5.19}$$

where θ is the tangential angle and s is the arc length. As observed, curvature has units of inverse distance and on most of our measurements we express in mm^{-1}.

The initial guess $\boldsymbol{u}_0$ and target control inputs $\boldsymbol{u}^*$ are approximated using curvature information from the model learning data.

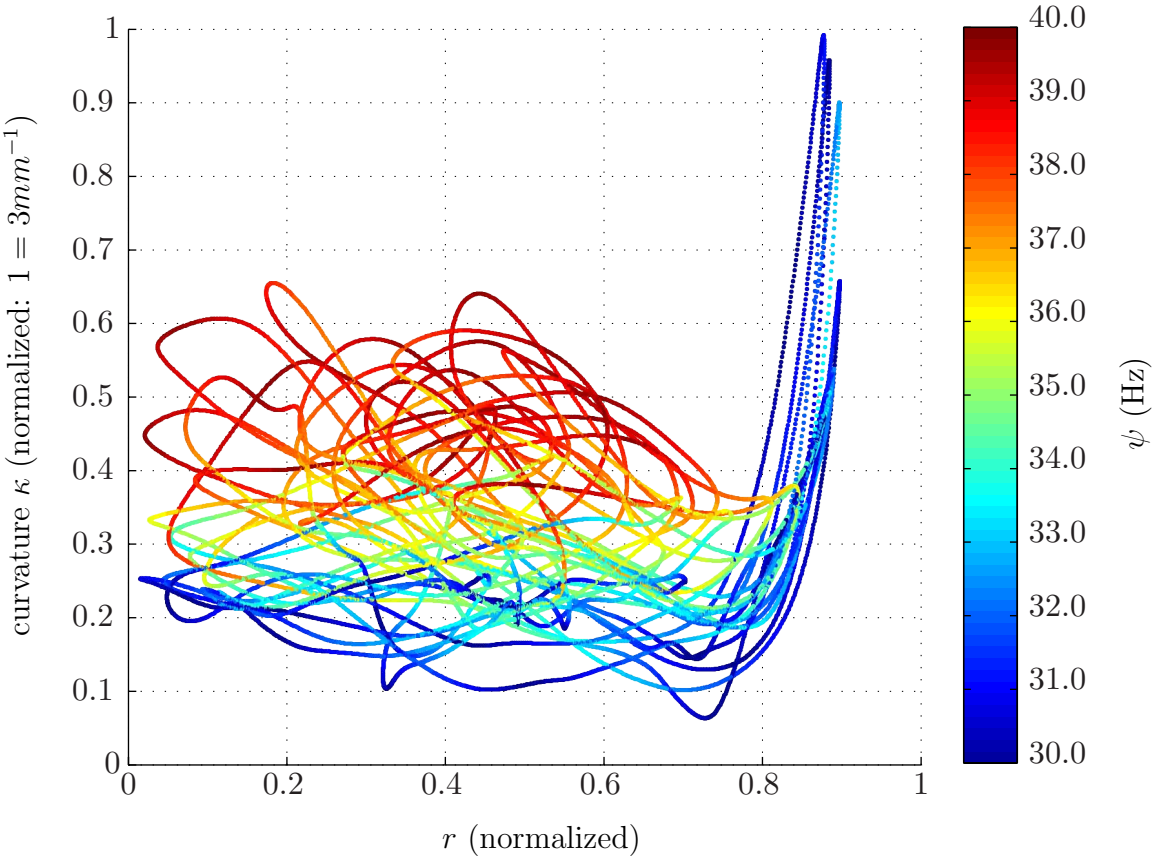

Figure 5.4: Curvature (κ) as a function of radius (r) and u. Each swimmer has a unique response to the input u, and have maximum and minimum curvatures which constrain feasible paths.

5.3.4 Receding Horizon

iLQR relies on continuous linearizations around a nominal trajectory, which is approximately feasible according to the system dynamics, and a computation of an LQR that will drive the tracking error to zero. This process is executed off-line and the resulting controller (or control sequence) used later on real-time.

However, on-line the controller might perform less optimally, if the system leaves the trajectory at which the iterations converged to. Several reasons for this to occur are perturbations, measurement and actuation errors, inaccurate model dynamics or an off-set on the initial state. This problem becomes more relevant when the open-loop control sequence $\bar{\boldsymbol{U}}_k^{(i)} = \{\bar{\boldsymbol{u}}_0^{(i)}, \bar{\boldsymbol{u}}_1^{(i)}, \ldots, \bar{\boldsymbol{u}}_K^{(i)}\}$ is directly used.

We apply the following extensions to solve this problem:

1. Update the LWPR model on real-time by feeding new state measurements as they become available and adjusting the λ parameter to set the learning rate.

2. Use a receding horizon: at the current time step k°, recompute the control problem by iLQR for $k = k^\circ, \ldots, K$.

Computing a feedback law for the duration of the complete motion primitive (from 7s to 30s) has an important limitation, since our simulator performance quickly starts degrading after 5s. We solve this problem by computing multiple times along the trajectory, similar to using a receding horizon, running the controller for a 1s long trajectory and recomputing at the end.

The swimmer is moving forward while a magnetic field is applied. Removing the field will cause the structure to rearrange and the dynamical behavior to change from the previous one. This is relevant since on our implementation, the time K^* it takes to compute the optimal control sequence is approximately 10% of its duration, during which we are not capable of doing feedback control, resulting on the initial conditions being off from the ones used on the beginning of the computation. The issue is solved by propagating the current state K^* steps on the simulator using a nominal control input and feeding the resulting state as initial condition.

By using this methodology, we are interested in tracking motion primitives in the form of circular paths. Starting with a particular circle, the user specifies the radius r^* and center location c_x, c_y, and the algorithm accomplishes the task by projecting the current point to the target path (closest point to the circle) and obtaining K target states $\boldsymbol{x}_k^*$. The iLQR controller is then computed as presented.

CHAPTER 6

HARDWARE EXPERIMENTS

This chapter presents the experimental procedure followed during a control trial. These trials consist of two stages: a model learning phase and a control phase, as shown on figure 4.3. The results are also discussed.

6.1 Experimental Procedure

For an experiment trial, <0.1g of 45μm nickel spheres are deposited over the water filled container (27.5mm water height) using a sieve (200-mesh) while no magnetic field is applied. We generate a sinusoidal magnetic field at $30Hz$ and allow the magnetic moment of the micro-particles to orient for the formation of chains followed by anti-ferromagnetically oriented segments, process which has been described in more detail on 3.1. Once a stable swimmer has been formed we linearly increase and decrease the frequency from $30Hz$ to $40Hz$ on $10s$ to allow further settlement of the segments. At this point the swimmer is symmetric and the water flow on both tails is balanced, resulting on a static structure. To induce an asymmetric structure we manually place a glass bead (1.5mm diameter Ni-Pd-Ni), the swimmer's head, close to one of the tails and position it slightly offset from the centerline. Offsetting the head to the right will produce a counterclockwise swimmer, while placing it to the left will produce a clockwise one. Further manipulation with a permanent magnet may be required to increase the segment particle density. Once a reliable swimmer has been formed we proceed to the model learning phase. We apply a triangular wave ranging from min to max frequencies, and record a minimum of $20,000$ swimmer states at $60Hz$. After completion, the time data is fed to our model learning algorithm which outputs the number of receptive fields K, curvature vs r, u and β plots and the MSE obtained with respect to the training data to ensure a similar performance to the values obtained during the tuning phase of LWPR. At this point the user may specify a target circle on the user interface and start the control process. The data stream during this process is recorded for later analysis.

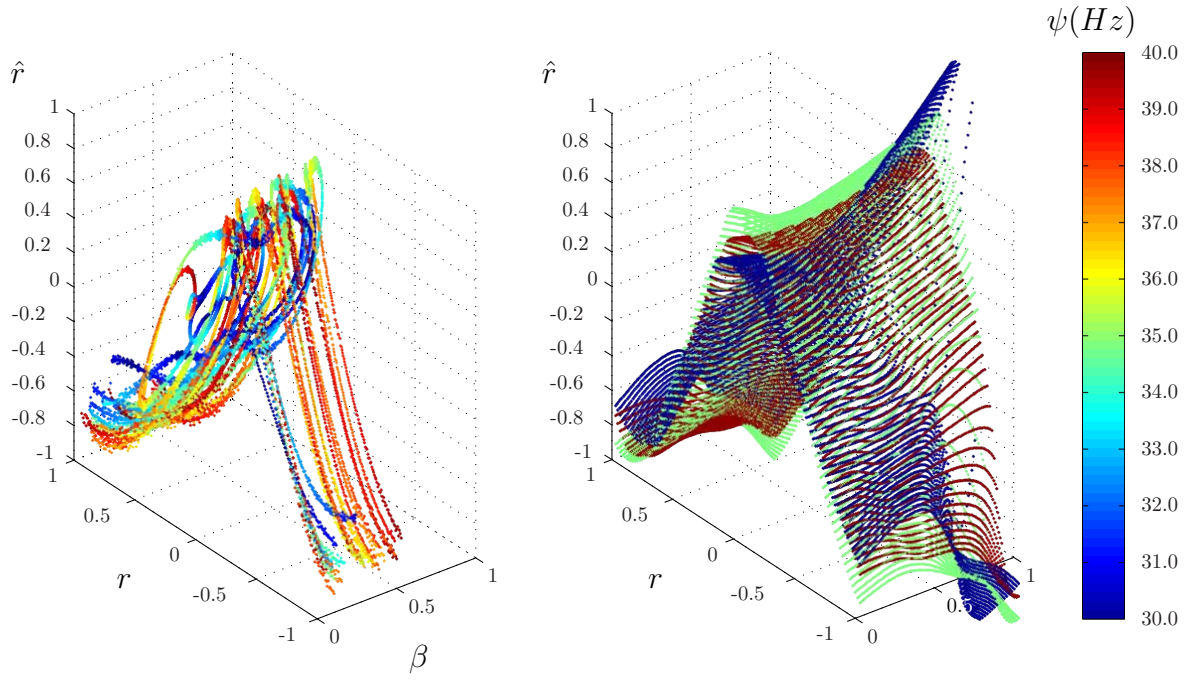

Figure 6.1: Normalized training data (left) and models (right) for $M_{\hat{r}}(r, \beta, u) \mapsto \hat{r}$.

6.2 Results

We conducted a series of experiments to assess the performance of our model learning algorithm and controller, and present two example swimmers following a circular motion primitive. We describe the results obtained for the model learning and motion primitive tracking phases.

1. *Model Learning:* The table 6.1 summarizes the performance of the models learned for both swimmers $S1$ and $S2$. As expected the nMSE obtained from the training data are lower that those for the test data, further more, the cross validations ($S1$ model vs $S2$ data, $S2$ model vs $S1$ data) present significant errors, up to 3x greater, than those from the test data. This shows the importance of learning a new model for each swimmer. As seen on Fig.6.4 the simulator is capable of predicting the spatial trajectory of the swimmer with low error up to a horizon of 5 s, giving us an important parameter for the design of the controller.

2. *Motion Primitive Tracking:* Our controller was able to reliably follow circle paths over multiple translations over a fixed range of curvatures. Figure 6.5 shows two example experiments compared to the swimmer trajectory used for learning the model.

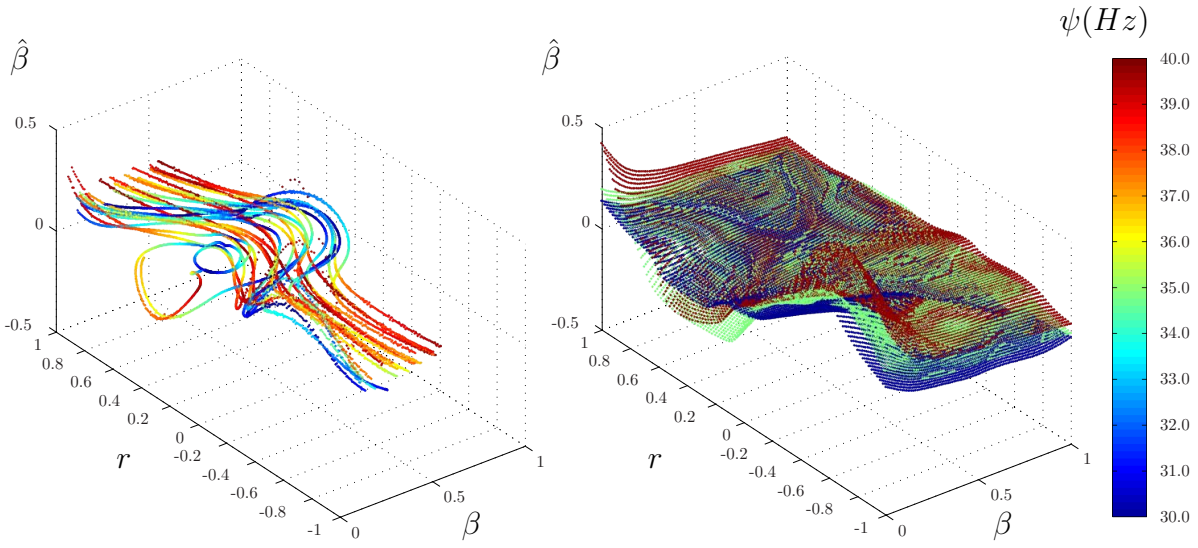

Figure 6.2: Normalized training data (left) and models (right) for $M_{\hat{\beta}}(r, \beta, u) \mapsto \hat{\beta}$.

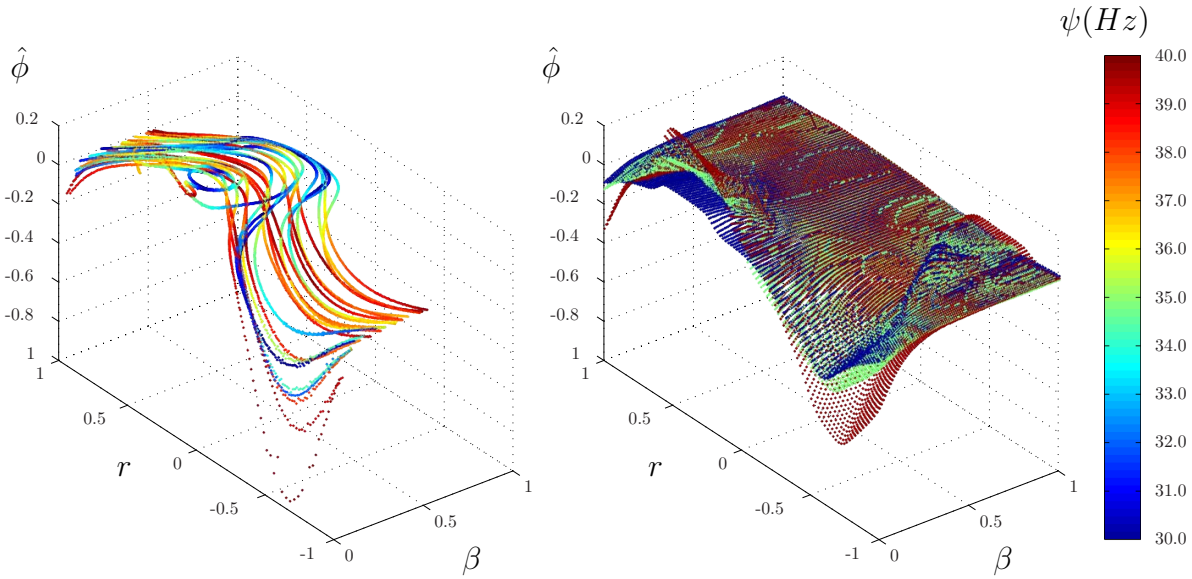

Figure 6.3: Normalized training data (left) and models (right) for $M_{\hat{\phi}}(r, \beta, u) \mapsto \hat{\phi}$.

		Model	
$\hat{r}$		S_1	S_2
S_1	train	4.52e-03	4.95e-02
	test	1.62e-02	3.90e-02
S_2	train	1.04e-01	1.18e-02
	test	1.26e-01	3.98e-02

Data			
$\hat{\beta}$		S_1	S_2
S_1	train	4.75e-02	1.35e+00
	test	2.94e-01	1.15e+00
S_2	train	4.68e-01	4.83e-02
	test	4.02e-01	8.51e-01

$\hat{\phi}$		S_1	S_2
S_1	train	3.31e-02	6.14e-01
	test	2.96e-01	5.16e-01
S_2	train	3.77e-01	2.19e-02
	test	1.38e-01	1.15e-01

Table 6.1: The table shows the normalized Mean Square Error (nMSE) between actual data and predicted values from LWPR for two swimmers, S_1 and S_2. Data is separated into a training set (80%) and a testing set (20%) a total of $26,490$ points where used for $S1$ and $20,791$ for S_2. The nMSE is computed for the training and test sets for each swimmer. Crossed cases are also considered.

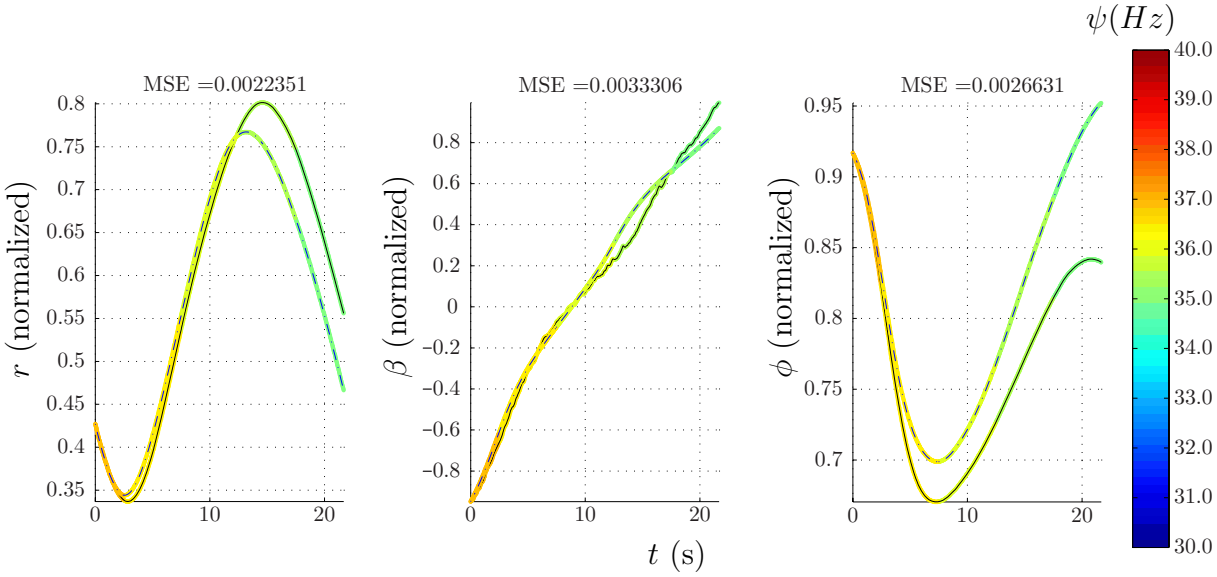

Figure 6.4: Predicted (dash) vs. actual (line) paths as a function of time. The simulator prediction has low error up to 5s. This value is used for calculating the controller's time horizon.

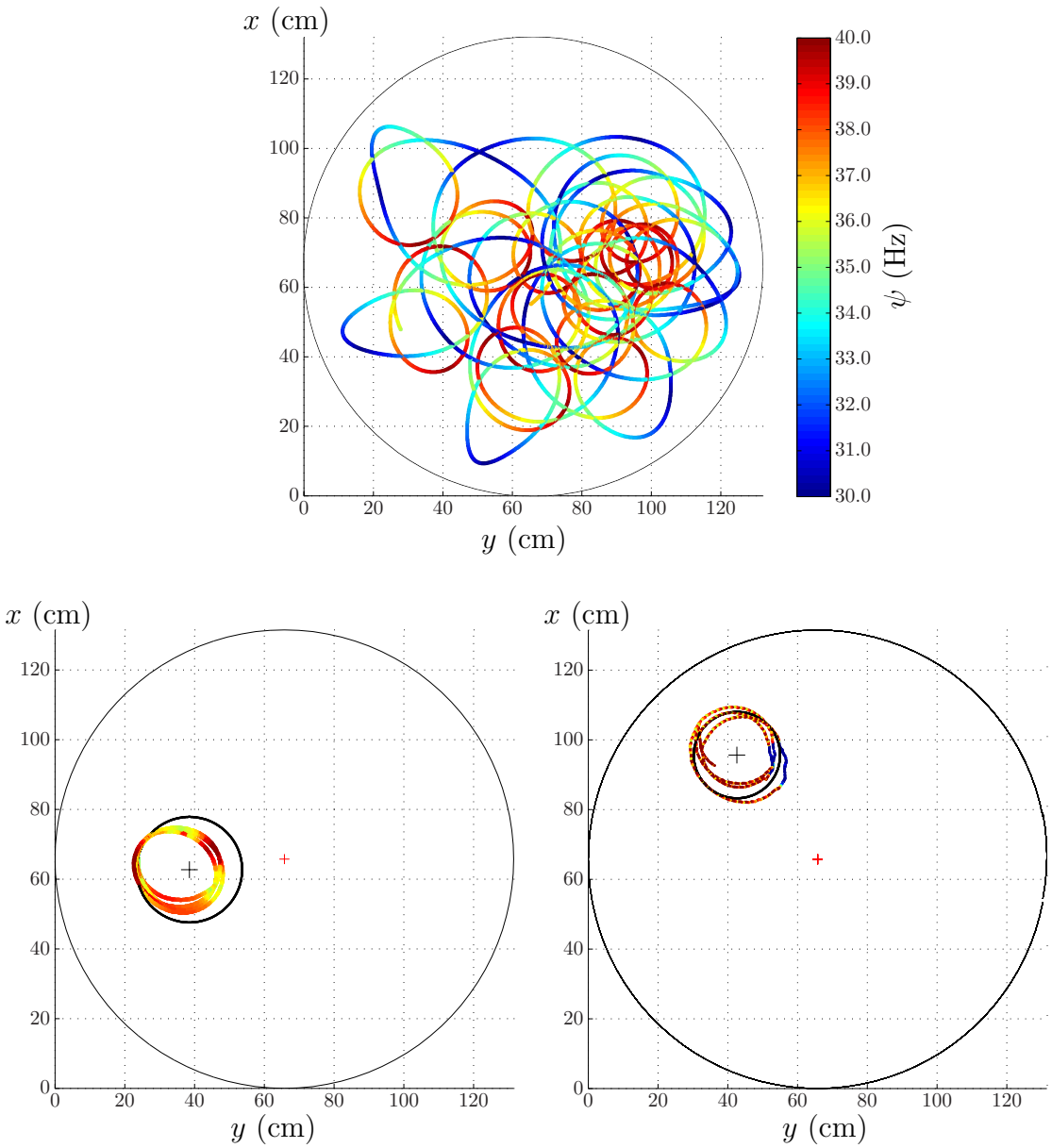

Figure 6.5: The top plot shows a swimmer trajectory under a ramp control input. This data is used to learn a model. The bottom plots show two experiments using the computed controller. The control task is to follow the black circle. To the left a controller with a receding horizon of 1s was used, to the right, one with a receding horizon of 0.2s

CHAPTER 7

CONCLUSIONS

In this thesis we examined the self-assembled magnetic structures introduced by Snezhko and Belkin [1, 2, 4]. These structures are of interest to the robotics community because they can be replicated with minimal hardware, are under-actuated yet controllable, and require model learning.

We presented a strategy using locally-weighted projection regression to learn dynamic models for self-assembled "swimmer" and a control policy based on iterative LQR to follow motion primitives. Through hardware experiments, we validated our dynamic models and implemented them to follow circular paths.

Future work should extend these results to point-to-point manipulation. For instance, to construct a feasible path from a start to goal location, as shown on figure 7.1. We could use our motion primitives as a set of inputs for a Rapidly-Exploring Random Tree [33, Chap. 14].

Similar methods could be employed for obstacle avoidance. Finally, while we have demonstrated that self-assembled swimmers have unique dynamic models, work remains before we can simultaneously control multiple swimmers.

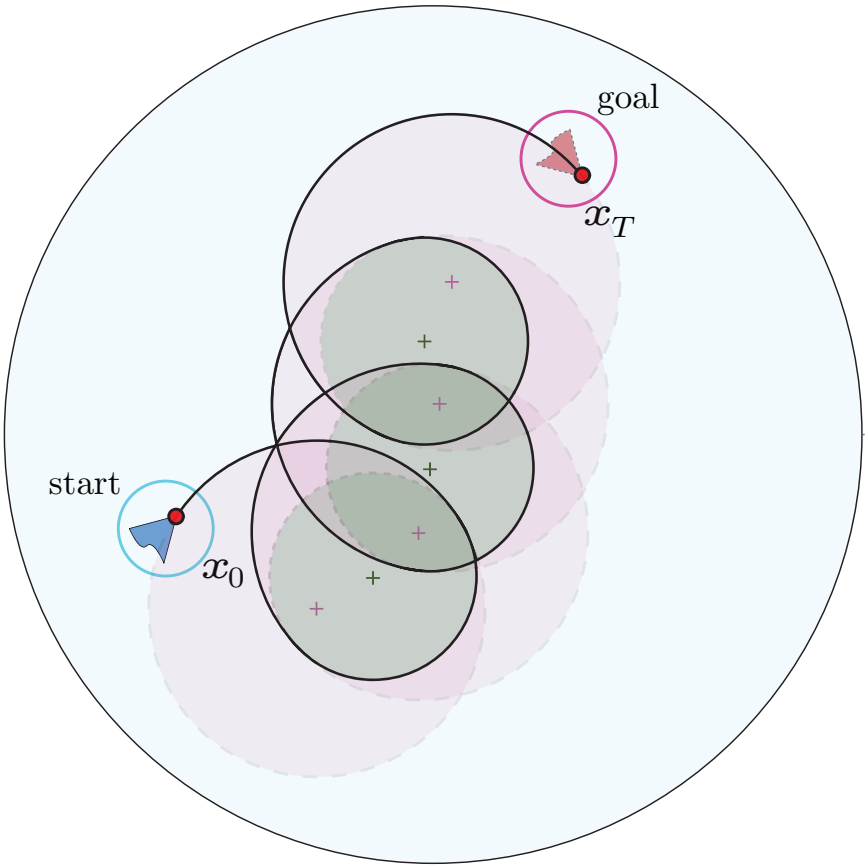

Figure 7.1: Future work will focus on extending these results for point-to-point manipulation by exploring a set of motion primitives that connect a start point $\boldsymbol{x}_0$ with a goal point $\boldsymbol{x}_T$.

REFERENCES

[1] A. Snezhko, M. Belkin, I. S. Aranson, and W.-K. Kwok, "Self-assembled magnetic surface swimmers." *Phys Rev Lett*, vol. 102, no. 11, p. 118103, 2009. [Online]. Available: http://www.biomedsearch.com/nih/Self-assembled-magnetic-surface-swimmers/19392241.html

[2] M. Belkin, A. Snezhko, I. S. Aranson, and W.-K. Kwok, "Driven magnetic particles on a fluid surface: pattern assisted surface flows." *Phys Rev Lett*, vol. 99, no. 15, p. 158301, 2007. [Online]. Available: http://www.biomedsearch.com/nih/Driven-magnetic-particles-fluid-surface/17995219.html

[3] M. Belkin, a. Snezhko, I. Aranson, and W.-K. Kwok, "Magnetically driven surface mixing," *Physical Review E*, vol. 80, no. 1, pp. 2–6, July 2009. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.80.011310

[4] A. Snezhko, I. S. Aranson, and W.-K. Kwok, "Surface wave assisted self-assembly of multidomain magnetic structures." *Phys Rev Lett*, vol. 96, no. 7, p. 78701, 2006. [Online]. Available: http://www.biomedsearch.com/nih/Surface-wave-assisted-self-assembly/16606148.html

[5] a. Snezhko, I. Aranson, and W.-K. Kwok, "Dynamic self-assembly of magnetic particles on the fluid interface: Surface-wave-mediated effective magnetic exchange," *Physical Review E*, vol. 73, no. 4, pp. 1–8, Apr. 2006. [Online]. Available: http://link.aps.org/doi/10.1103/PhysRevE.73.041306

[6] S. Vijayakumar, A. D'Souza, and S. Schaal, "Incremental online learning in high dimensions." *Neural computation*, vol. 17, no. 12, pp. 2602–34, Dec. 2005. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/16212764

[7] W. Li and E. Todorov, "Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems systems," *Int. Conf. on Inf. in Cont., Auto., and Rob.*, pp. 1–8, 2004.

[8] P. Abbeel, a. Coates, and a. Y. Ng, "Autonomous Helicopter Aerobatics through Apprenticeship Learning," *The International Journal of Robotics Research*, vol. 29, no. 13, pp. 1608–1639, June 2010. [Online]. Available: http://ijr.sagepub.com/cgi/doi/10.1177/0278364910371999

[9] D. A. LaVan, D. M. Lynn, and R. Langer, "Moving smaller in drug discovery and delivery," *Nature Reviews Drug Discovery*, vol. 1, pp. 77–83, 2002. [Online]. Available: http://direct.bl.uk/bld/PlaceOrder.do?UIN=111646375\&ETOC=RN\&from=searchengine

[10] M. Hashizume and K. Tsugawa, "Robotic Surgery and Cancer: the Present State, Problems and Future Vision," *Japanese Journal of Clinical Oncology*, vol. 34, no. 5, pp. 227–237, 2004. [Online]. Available: http://jjco.oxfordjournals.org/content/34/5/227.abstract

[11] E. Kreyszig, *Differential Geometry*, 1st ed. Dover Publications, Inc., 1991.

[12] G. M. Whitesides and B. Grzybowski, "Self-Assembly at All Scales," *Science*, vol. 295, no. 5564, pp. 2418–2421, 2002. [Online]. Available: http://www.sciencemag.org/content/295/5564/2418.abstract

[13] T. D. Murphey, J. Bernheisel, D. Choi, and K. M. Lynch, "An example of parts handling and self-assembly using stable limit sets," in *Int. Conf. Int. Rob. Sys.*, Aug. 2005, pp. 1624–1629.

[14] R. Gross, M. Bonani, F. Mondada, and M. Dorigo, "Autonomous Self-Assembly in Swarm-Bots," *IEEE Trans. Robot.*, vol. 22, no. 6, pp. 1115–1130, Dec. 2006.

[15] S. S. Sudo Sudo and T. Honda, "Magnetic Swimming Mechanism in a Viscous Liquid," *Journal of Intelligent Material Systems and Structures*, vol. 17, no. 8-9, pp. 729–736, 2006.

[16] J. J. Abbott, K. E. Peyer, M. C. Lagomarsino, L. Zhang, L. X. Dong, I. K. Kaliakatsos, and B. J. Nelson, "How Should Microrobots Swim?" *Int. J. Rob. Res.*, July 2009.

[17] L. Zhang, J. J. Abbott, L. X. Dong, B. E. Kratochvil, D. J. Bell, and B. J. Nelson, "Artificial Bacterial Flagella: Fabrication and Magnetic Control," *Applied Physics Letters*, vol. 94, no. 6, Feb. 2009.

[18] S. Floyd, E. Diller, C. Pawashe, and M. Sitti, "Control methodologies for a heterogeneous group of untethered magnetic micro-robots," *I. J. Robotic Res.*, vol. 30, no. 13, pp. 1553–1565, Nov. 2011.

[19] E. Diller, S. Floyd, C. Pawashe, and M. Sitti, "Control of Multiple Heterogeneous Magnetic Microrobots in Two Dimensions on Nonspecialized Surfaces," *IEEE Trans. Robot.*, vol. 28, no. 1, pp. 172–182, Feb. 2012.

[20] A. J. Smola and B. Schlkopf, "A tutorial on support vector regression," *Statistics and Computing*, vol. 14, 2004.

[21] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. MIT Press, 2006.

[22] C. Atkeson and A. Moore, "Locally weighted learning for control," *Artificial Intelligence Review*, pp. 1–33, 1997. [Online]. Available: http://www.springerlink.com/index/N310JR14258M3373.pdf

[23] M. Kalakrishnan, J. Buchli, P. Pastor, and S. Schaal, "Learning locomotion over rough terrain using terrain templates," *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 167–172, Oct. 2009. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5354701

[24] J. Morimoto and G. Zeglin, "Minimax differential dynamic programming: Application to a biped walking robot," *Robots and Systems, 2003*, no. i, pp. 1–6, 2003. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1248926

[25] J. Peters and S. Schaal, "Learning to Control in Operational Space," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 197–212, Feb. 2008. [Online]. Available: http://ijr.sagepub.com/cgi/doi/10.1177/0278364907087548

[26] S. Vijayakumar and A. D'Souza, "LWPR: A scalable method for incremental online learning in high dimensions," *Neural Computation*, pp. 1–34, 2005. [Online]. Available: http://www.era.lib.ed.ac.uk/handle/1842/3705

[27] S. Vijayakumar, "Locally weighted projection regression: An O (n) algorithm for incremental real time learning in high dimensional space," *International Conference on Machine Learning*, vol. 1086, pp. 1079–1086, 2000. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.63.4252\&rep=rep1\&type=pdf

[28] S. Schaal and C. Atkenson, "Receptive field weighted regression," *Processing Laboratories, Tech. Rep. TR-H-209*, 1997. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.56.6336\&rep=rep1\&type=pdf

[29] S. Schaal, C. Atkenson, and S. Vijayakumar, "Scalable techniques from nonparametric statistics for real time robot learning," *Applied Intelligence*, vol. 16, 2002.

[30] S. Schaal and C. Atkeson, "Constructive incremental learning from only local information," *Neural Computation*, vol. 10, 1998.

[31] S. Klanke, S. Vijayakumar, and S. Schaal, "A Library for Locally Weighted Projection Regression," *Journal of Machine Learning Research*, vol. 9, pp. 623–626, 2008.

[32] B. Anderson and J. Moore, *Optimal Control Linear Quadratic Methods*. Englewood Cliffs, NJ: Prentice Hall International, Inc., 1989.

[33] S. M. LaValle, *Planning Algorithms*. Cambridge University Press, 2006.